Improving reliability of mobile manipulators against unknown external faults

Naveed Akhtar

Publisher: Dean Prof. Dr. Wolfgang Heiden

University of Applied Sciences Bonn-Rhein-Sieg,
Department of Computer Science

Sankt Augustin, Germany

March 2012

Technical Report 03-2012

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

ISSN 1869-5272

Copyright © 2012, by the author(s). All rights reserved. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Das Urheberrecht des Autors bzw. der Autoren ist unveräußerlich. Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Das Werk kann innerhalb der engen Grenzen des Urheberrechtsgesetzes (UrhG), *German copyright law*, genutzt werden. Jede weitergehende Nutzung regelt obiger englischsprachiger Copyright-Vermerk. Die Nutzung des Werkes außerhalb des UrhG und des obigen Copyright-Vermerks ist unzulässig und strafbar.

Abstract

A robot (e.g. mobile manipulator) that interacts with its environment to perform its tasks, often faces situations in which it is unable to achieve its goals despite perfect functioning of its sensors and actuators. These situations occur when the behavior of the object(s) manipulated by the robot deviates from its expected course because of unforeseeable circumstances. These deviations are experienced by the robot as *unknown external faults*. In this work we present an approach that increases reliability of mobile manipulators against the unknown external faults. This approach focuses on the actions of manipulators which involve releasing of an object. The proposed approach, which is triggered after detection of a fault, is formulated as a three-step scheme that takes a definition of a planning operator and an example simulation as its inputs. The planning operator corresponds to the action that fails because of the fault occurrence, whereas the example simulation shows the desired/expected behavior of the objects for the same action. In its first step, the scheme finds a description of the expected behavior of the objects in terms of logical atoms (i.e. *description vocabulary*). The description of the simulation is used by the second step to find limits of the parameters of the manipulated object. These parameters are the variables that define the releasing state of the object. Using randomly chosen values of the parameters within these limits, this step creates different examples of the releasing state of the object. Each one of these examples is labelled as desired or undesired according to the behavior exhibited by the object (in the simulation), when the object is released in the state corresponded by the example. The description vocabulary is also used in labeling the examples autonomously. In the third step, an algorithm (i.e. N-Bins) uses the labelled examples to suggest the state for the object in which releasing it avoids the occurrence of *unknown external faults*.

The proposed N-Bins algorithm can also be used for binary classification problems. Therefore, in our experiments with the proposed approach we also test its prediction ability along with the analysis of the results of our approach. The results show that under the circumstances peculiar to our approach, N-Bins algorithm shows reasonable prediction accuracy where other state of the art classification algorithms fail to do so. Thus, N-Bins also extends the ability of a robot to predict the behavior of the object to avoid *unknown external faults*. In this work we use simulation environment OPENRave that uses physics engine ODE to simulate the dynamics of rigid bodies.

Acknowlegements

I would like to thank Dr. Paul G. Ploeger, Dr. Alexander Asteroth and Ms. Anastassia Kuestenmacher for being my advisors for this work. I am also very grateful to Ms. Iman Awad for her invaluable advices and discussions throughout the work.

I am grateful to my colleagues Sven, Matthias, Pinaki, Marcel and Chris for their technical help and useful discussions. I am especially grateful to Sven and Matthias because of all the help over the complete duration of my Master's degree which also enabled me to complete this work successfully.

I am grateful to my parents and my siblings for their continuous moral support and never ending care. I am also thankful to Higher Education Commission (HEC) of Pakistan and Deutscher Akademischer Austausch Dienst (DAAD) for the financial support.

Contents

List of Tables

List of Figures

Abbreviations

AI	Artificial Intelligence
ANN or NN	Artificial Neural Networks
CAD	Computer Aided Design
CCW	Counter Clock Wise
CW	Clock Wise
DT or D-Tree	Decision Tree
FOL	First Order Logic
HTN	Hierarchical Task Network
JEPD	Jointly Exhaustive Pairwise Disjoint
K-NN	K-Nearest Neighbors
KB	Knowledge Base
ML	Machine Learning
MTBF	Mean Time Between Failure
ODE	Open Dynamic Engine
OpenRAVE	Open Robotics Automation Virtual Environment
RCC	Region Connection Calculus
QR	Qualitative reasoning
QSR	Qualitative Spatial Reasoning
SVM	Support Vector Machines

1 Introduction

Faults can occur even in the most carefully designed systems. Normally, these faults occur because of malfunctioning/failure of a system's internal components. However, for a system that interacts with its environment, malfunctioning/failure of internal components is not the only source of occurrences of faults. Such a system also faces *situations in its environment*, which prohibit it from successfully achieving its goals. These situations occur despite perfect functioning of the system's internal components. These situations are usually unforeseeable at the development phase of a system and remain *unknown* until their occurrences. Under conventional definitions of systems, these situations are *external* to the systems. Furthermore, they manifest themselves as deviations of behavior or properties of entities (in the environment of a system) from their expected values. Therefore, these situations are referred as *unknown external faults* (Akhtar and Kuestenmacher [2011]).

In this work we propose an approach for increasing reliability of mobile manipulators against *unknown external faults*. The most common task for mobile manipulators is picking and placing of objects in their environment. While performing this task a mobile manipulator is vulnerable to *unknown external faults* because of its interaction with the objects. This work focuses on enabling manipulators to perform the action of 'placing' objects in a reliable manner. The approach proposed in this work accomplishes this goal by finding an appropriate state of the manipulated object. Releasing the object in this state always results in the expected/desired behavior of the object after its release. In order to find the appropriate state, the approach makes use of a simulation process. This process simulates the dynamics of the object and generates different examples of the behavior of the object after the action of release. Based on the desirability of the behavior of the object in each example, the approach estimates the appropriate state of the object that avoids the occurrence of *unknown external faults*.

The approach proposed in this work is formulated as a three-step scheme that can be used for plan based robotic systems. This scheme assumes that the robot (i.e. the mobile manipulator) is able to detect the occurrence of the fault at the planning level by monitoring the *effects* of an executed action. It also assumes availability of a simulation that shows an example of the expected behavior of the manipulated object for the case of successful completion of the executed action. This simulation can always be made available for a planning operator (i.e. an action) because a plan based robotic system assumes availability of the model of its environment. In its first step, the scheme uses the simulation of the sample behavior of the object to find a logical description of the

expected behavior. This description is given by two logical sentences which depict the states of the object at the start and at the end of the simulation. The second step of the scheme uses the description of the simulation to find the limits of parameters of the object. These parameters are the variables that define the releasing state of the object.

In order to find the appropriate state of the object, the proposed scheme uses the limits of the parameters to create different examples of the releasing state of the object. Each one of these examples is labelled as desired or undesired according to the behavior exhibited by the object (in the simulation), when it is released in the state corresponded by the example. The scheme uses the description of the sample behavior in autonomous labeling of the examples. In the third step of the scheme the labelled examples are used by an algorithm, called N-Bins. This algorithm estimates the best and the worst states of releasing the object, within the limits of the parameters found by the approach. Once this algorithm is supplied with the labelled examples, it also develops the ability to predict that whether or not a given releasing state of the object is a desired state to accomplish the action successfully. The scheme proposes to incorporate the knowledge of the appropriate state of the object and the ability of prediction of the said algorithm in the *preconditions* of the planning operator that detects the occurred *unknown external fault*. If a robot performs the action by satisfying the modified preconditions then it can avoid the occurrence of the fault because these preconditions already ensure that the object shows the desired behavior in the action.

The above mentioned three steps of the scheme use many small techniques and exploit many small facts to achieve their goals. Therefore, in this thesis we mainly focus on the descriptive explanation of the basic approach and formulate the parts of the approach as algorithms only where necessary. The explanation of the approach is also distributed in different chapters of the thesis for the ease of understanding. This work is among the first few works[1] which treats the issue of *unknown external faults* as a primary research problem. This makes the scope of this work very broad. Therefore, the basic scheme of the approach is kept very general. Furthermore, the tools used by the scheme (e.g. N-Bins algorithm, logical expressions to find the simulation description) are also developed systematically for the sake of extensibility.

In order to evaluate our approach we conduct different experiments in which we find the (approximate) best (and the worst) way(s) of releasing different objects over other objects. In these experiments we also evaluate the accuracies of predictions of N-Bins algorithm and compare them with the accuracies of other state of the art (machine learning) algorithms which posses the same ability of prediction. Results of the experiments show that using

[1] According to the best of our knowledge.

the proposed approach it is possible to estimate the (approximate) best (and the worst) way(s) of performing the action. Furthermore, the results show that along with another state of the art algorithm (i.e. artificial neural network), N-Bins can be used to predict the behavior of the objects with reasonable accuracy with a very less number of examples generated from the simulation process.

This thesis is organized in nine chapters. The next chapter of the thesis gives the background knowledge required to comprehend the work in this thesis. Chapter 3 gives the state of the art of the area of fault tolerance and diagnosis in robotics. In chapter 4 we give a brief explanation about the *unknown external faults* and raise important observations about their nature. These observations form the basis of our approach, which is explained in chapter 5. The proposed approach makes use of logical relations (i.e. predicates) to find the description of the simulation. We refer to the relations collectively as *description vocabulary*. Chapter 6 explains the definitions of the relations used in description vocabulary. In chapter 7 we give the results of applying our approach to different objects. This chapter also gives the analysis of the results. Chapter 8 gives brief reviews of some of the works related to our approach. The conclusion of the thesis and future directions of the work are stated in chapter 9.

2 Background

This chapter gives a brief review of the concepts used in this work. The aim of this chapter is only to familiarize the reader with the terms and concepts used in this work. Therefore, we do not provide complete details about the concepts or about the relevant fields. Where necessary, the chapter provides references to the literature that can be useful for the details. However, understanding of such details is not mandatory to comprehend the approach developed in this work.

2.1 Machine Learning (ML)

Machine Learning (ML) is an area in Artificial Intelligence (AI) that studies the algorithms that make a computing machine to evolve with experience with the help of computer programs. According to Tom M. Mitchell, " a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at the tasks in T, as measured by P, improves with experience E" (Mitchell [1997]). The tasks in the area of ML mostly consist of learning models for systems, such that the models are able to predict the behavior of the systems whenever they are provided with the necessary inputs. Another important category of the problems in ML is to find models that are able to *classify* some unseen data into different classes based on the available data. A computer program or the technique that finds the above mentioned models, is termed as a *learning algorithm*. The learning algorithm is provided with the experience in the form of data or the examples of inputs and the corresponding response of the system. These examples are called the *training examples*. In the process of learning, the performance measure of a learning algorithm is mostly based on the prediction or classification accuracy of the model that is being learned by the algorithm.

The model that is learned by a learning algorithm can be represented as a *hypothesis* that is found by the algorithm after its exposure to the training examples. Once a hypothesis is learned by the algorithm its accuracy can be evaluated by exposing it to some unseen data, that consists of *test examples*. Normally, before a learning algorithm can be used to find a hypothesis, it needs to be provided with certain parameters. These parameters adjust the behavior of the algorithm in the learning process. For instance, some of the learning algorithms require the value of a parameter known as *learning rate*, that is mainly responsible for the time required by an algorithm to learn a hypothesis. In order to find a better hypothesis, it is important to adjust the parameters of a learning algorithm to

better values. These values can be found by the process of *cross validation* of a learned hypothesis. In this process, some examples are set aside before the start of learning. These example are used to evaluate the accuracy of the hypothesis that is learned with a particular set of values of the parameters of the learning algorithm. Once a hypothesis is learned, the process is repeated with new values of the parameters. If the performance of the new hypothesis on the *cross validation set* of examples is better than earlier, then the new values of the parameters are retained. This process of cross-validation is repeated until some desired performance is achieved. Finally, the performance of the algorithm is evaluated by exposing it to the *test examples*.

In the above explanation of the terms related to ML, an example (training or test) is considered to be consisting of following two parts:

- A set or a vector of *features*, which represents the input variables for a system or a classifier.
- A variable or a set of variables that corresponds to the output response of the system or the classifier for the given input *feature* vector.

The type of learning that needs the output response of the 'to be learned' model for each training example, is termed as *supervised learning*. Other major types of learning in ML are *unsupervised learning* and *reinforcement learning*. In this section we do not include any discussion on these types of learning because of their irrelevance to this work[1].

For the problems in which the goal is to *classify* an input feature vector (i.e. an example), the output variable is termed as the *label* of the class represented by the example. There is a specific type of classification problems in which the labels can take on only two values. This type is called *binary-classification* or *concept learning*. Examples of ML algorithms that *can* be used for binary classification problems include, *decision trees, artificial neural networks, support vector machines* and *k-nearest neighbors* etc. In binary classification problems, examples (training and test) are also termed as *positive* and *negative* examples depending upon the labels of the examples. Once a classifier is trained, the positive and negative examples in the set of test examples can be used to evaluate performance metrics which are more reliable than calculating mere accuracy of the classification. Two of these metrics are called *precision* and *recall* of the learning algorithm. Equations 2.1 and 2.2 give the formulae of these metrics. The examples referred by these formulae are the test examples. In these equations, *true positive* examples are those which are correctly classified as positive by the classifier, whereas *false positive* examples are the ones which are falsely classified as positive.

[1]Readers interested in unsupervised or reinforcement learning or details of the concepts described in this section, can find useful discussion in Mitchell [1997].

$$Precision = \frac{\#\text{ of true positive examples}}{\#\text{ of true positive examples} + \#\text{ of false positive examples}} \tag{2.1}$$

$$Recall = \frac{\#\text{ of true positive examples}}{\#\text{ of actual positive examples}} \tag{2.2}$$

2.2 First-Order Logic (FOL)

First Order Logic (FOL) is a representation language that uses logical symbols to encode knowledge. It is also known as predicate calculus or predicate logic. In knowledge representation FOL either subsumes other representation languages (e.g. propositional logic) or forms the foundation of such languages (e.g. higher order logics) (Russell and Norvig [2002]). In FOL knowledge is represented in the form of expressions. These expressions consist of *variables* (e.g. `x, y, obj`), *constants* (e.g. `Die, Table`[2]), *function* symbols (e.g. `height(obj)`) and *predicate* symbols (e.g. `Above(obj1, obj2)`). A *predicate* symbol evaluates to *true* or *false* values when its argument gets instantiated with constant symbols. On the other hand, a *function* symbol refers to another *constant* value upon the instantiation of its argument. For example, the predicate `Above(Die, Table)` may evaluate to *true* under some intended *interpretation* if an object `Die` is above another object `Table`. Whereas, `height(Table)` may refer to a constant value representing the actual height of the object `Table`. It is conventional in FOL that the first letter for the name of a function is kept small, whereas the name of a predicate starts with a capital letter. We also follow the same convention in this work. The logical symbols that evaluate to *true* or *false* values are also termed as *atoms* in FOL.

FOL uses logical connectives (i.e. 'or' $\vee$, 'and' $\wedge$, 'not' $\neg$, 'implication' $\Rightarrow$ and 'double implication' $\Leftrightarrow$) to connect the above mentioned symbols into *sentences*. Each variable in such sentences is quantified using *quantifiers* (i.e. 'for all' $\forall$ and 'there exists' $\exists$). Below (expression 2.3) is an example of a sentence in FOL. According to our interpretation, this sentence gives the definition of a concept of movability of an `object`. That is, "an `object` is *movable* if and only if there exists at least one instant `t` in which the `object` is *not stationary*". The condition stated on the right hand side of '$\Leftrightarrow$' sign is called the *body* of the sentence. In the sentence 2.3, if the body of the sentence is true for an `object` then `Movable/1` is also evaluated to *true* for that `object`.

$$\forall_{object} \quad \texttt{Movable(object)} \Leftrightarrow \exists_t[\neg\texttt{Stationary(object, t)}]. \tag{2.3}$$

[2]In FOL it is conventional to use a small letter for the first alphabet in the name of a *variable* and a capital letter for the first alphabet in the name of a *constant*. In this work we also follow the same convention in FOL expressions. However, in the textual explanations we sometimes also use smaller letters for the first letter of the names of *constants*.

It is possible to perform *inferencing* over the collection of sentences like above. Any such collection of sentences is termed as *Knowledge Base* (KB). FOL performs inferencing over KB using *inferencing rules* (e.g. unification, resolution) which are applied to the logical symbols comprising the sentences. This means that arithmetic operations are not a part of logical inferencing. However, it is possible to represent arithmetic operation in FOL by abstracting them as logical symbols. For instance, an operation of summation can be abstracted as an `Add(x,y,z)` *predicate* which is true when `z` is equal to the sum of `x` and `y`, and false otherwise.

Although FOL possesses great representation power, but like any other logical language when it is used to encode purely qualitative information, there is a loss of information in the process of abstraction. For instance Bratko [2001] shows that a simple arithmetic operation of summation can become non-deterministic when it is abstracted as a *predicate*. Loss of information at the hands of abstraction renders inferencing in FOL less suitable for the applications where arithmetic operations are necessary. However, representation capabilities of FOL still make it a powerful tool for encoding knowledge in a compact manner.

2.3 Automated planning

[3]For an agent (e.g. a robot), planning is the process of deliberation that enables it to choose and organize its actions based on their outcomes. The area of AI that studies this process computationally is termed as automated planning. It is possible to formalize a problem of automated planning with the help of a conceptual model for a system under consideration. This model is represented by a state-transition-system Σ which is a 4-tuple (S, A, E, γ), where:

- $S = \{s_1, s_2, ...\}$ is a finite set of *states* of the system;
- $A = \{a_1, a_2, ...\}$ is a finite set of *actions* that can be performed;
- $E = \{e_1, e_2, ...\}$ is a finite set of *events*; and
- $\gamma : S \times A \times E \rightarrow 2^S$ is a *state transition function.*

In the above description of Σ, *actions* and *events* cause transitions in the *states* of the system. For a plan executor, *actions* correspond to the controlled transitions, whereas *events* correspond to the uncontrolled transitions caused by the dynamics of the system. In order to formalize a planning problem, some restrictive assumptions can be made on

[3]Contents in this section are mainly based on the description of automated planning in Ghallab et al. [2004].

the conceptual model. By relaxing these assumptions different models of the system can be obtained. These assumptions are as following:

1. Σ *(i.e. the system) has only finite possible states.*
2. Σ *is fully observable.* This means that complete knowledge about a state of Σ is available.
3. Σ *is deterministic.* That is, an *action* or an *event* can bring Σ to only a single other state.
4. Σ *is static.* This implies that E is an empty set.
5. *The goals are restricted.* That means, the goals of Σ can only be explicit states or a set of states.
6. *A solution to the planning problem is a sequence of actions.* This sequence is linearly ordered and finite.
7. *Actions and events have no duration.* That is, they are performed instantly.
8. *Planning is performed offline.*

2.3.1 Classical planning

A state-transition system that meets all the restrictive assumptions mentioned above is called a restricted state-transition system. Planning for a restricted state-transition system is referred as *classical planning*. The planning problem for the restricted state-transition system is defined as a triple $\Pi = (\Sigma, s_0, g)$, where:

- s_0 is the initial state of the system Σ; and
- g corresponds to the goal state(s).

A solution to Π is a sequence of actions $(a_1, a_2, ..., a_n)$ that corresponds to the states $(s_0, s_1, ..., s_n)$ such that $s_1 = \gamma(s_0, a_1), ..., s_n = \gamma(s_{n-1}, a_n)$, and s_n is the goal state or a set of goal states.

Classical representation

Classical representation is one of the ways of formally representing a classical planning problem. In classical representation states of the system are represented by FOL atoms that are either *true* or *false* according to some interpretation. Actions are represented by *planning operators*, that cause changes in the truth values of the logical atoms. A *planning operator* is formally represented as a triple $o = (name(o), preconditions(o), effects(o))$, where:

- $name(o)$ is a unique syntactic expression representing the name of the planning operator.
- $preconditions(o)$ are the logical atoms or their negations that represent the conditions that must be satisfied before an action can be performed.
- $effects(o)$ are the logical atoms or their negations that are satisfied after an action has been completed.

```
Put-on(object-1, object-2).
preconditions: Holding(agent, object-1), ¬On(object-1, object-2),
               Empty(object-2).
effects:       On(object-1, object-2), ¬Holding(agent, object-1).
```

Figure 2.1: An example of planning operator.

Above is an example of a *planning operator* for an action of putting an object on another object. Here, the predicate `Holding/2` is true when the `agent` is holding `object-1` in its manipulator. The predicate `On/2` is true when `object-1` is on `object-2`, and `Empty/1` is true when the top surface of `object-2` is empty. In order to execute a plan that includes the action of putting an object on another object, the `Put-on` operator is instantiated with the relevant objects. During the execution of a plan, this instantiation can only take place when a particular state of the system satisfies the *preconditions* of `Put-on`. If the action is executed in such a state, its successful completion results in a state that satisfies the *effects* of `Put-on`. That is, the system is able to transit from a state where it is holding `object-1` to a state where `object-1` is placed on `object-2` and the `agent` is no longer holding it.

It can be noticed in the above description that in *classical representation* states of a system are merely conjunctions of predicates. A change in a state corresponds to the change in the truth values of the relevant predicates and this change occurs only upon the execution of actions. Execution of the actions only takes place when relevant *planning operators* get instantiated into actions. This instantiation is performed according to the sequence found by the *planner*. Upon a complete execution of the sequence of actions (i.e. the plan) the system is able to achieve its *goal state*.

2.3.2 Hierarchical Task Network (HTN) planning

Like *classical planning, hierarchical task network* (HTN) *planning* also represents the state of the world in terms of logical atoms. Furthermore, HTN planning also considers the transition between the states to be deterministic. However, there is a major difference

between HTN planning and classical planning. That is, instead of finding a goal state the objective of HTN planning is to perform a set of *tasks*, where a single task may require execution of many actions. Each one of these actions are carried out by instantiating the *planning operators*.

An HTN planner plans by breaking the tasks into subtasks. This process of breaking down a task is performed according to the *methods* which provide the prescription of decomposition of the tasks. In the process of planning the *non-primitive* tasks are recursively decomposed into subtasks until *primitive tasks* are reached. These *primitive tasks* can be performed directly with the help of *planning operators* just like in the case of *classical planning*.

HTN planning is one of the most popular planning techniques in AI and there are a lot of details that need to be understood to comprehend this planning technique completely[4]. However, knowledge of these details is not necessary for understanding the approach developed in this work. The major issue that must be kept in mind to understand our approach is that, in HTN planning the primitive tasks can be performed directly using the *planning operators* just like the one shown in section 2.3.1. In other words, execution of a single action is achieved exactly in the same manner for both the HTN planning and the classical planning.

2.4 Qualitative Spatial Reasoning (QSR)

Qualitative reasoning (QR) is an area of AI which creates representations for continuous aspects of the world which support reasoning with very little information (Forbus [2003]). *Qualitative spatial reasoning* (QSR) is a subarea within QR that addresses reasoning about spatial aspects of an environment. Unlike other aspects of nature (e.g. time, quantity) space is multidimensional. This makes it hard to find a purely qualitative representation of space that can be used in qualitative reasoning in an extensible manner. This fact is noted as *poverty conjecture* by Forbus et al. [1991], where the authors state that *"no general purpose, purely qualitative representation of spatial properties exists"*. Cohn and Hazarika [2001] also seconds this notion.

In order to formalize the knowledge about space one has to consider many of its aspects. For example, any spatial reasoning system needs to consider the ontology, spatial relations, mereology (i.e. part-hood), topology, mereo-topology, directions, orientations and other such aspects related to the spatial knowledge of the domain. A detailed discussion

[4]Interested readers can find a good description of HTN planning in chapter 11 of Ghallab et al. [2004].

on these aspects and their importance in qualitative spatial reasoning can be found in Cohn and Hazarika [2001]. We do not include such a discussion here because this work only exploits the systematic representation of spatial knowledge used in QSR. Detailed understanding of QSR is not compulsory for understanding the approach developed in this work.

2.4.1 RCC-8

In the area of qualitative spatial representation and reasoning, Region Connection Calculus (RCC) (Randell et al. [1992]) is one the most widely used formalisms. RCC is a many-sorted first-order axiomatisation of spatial relations based on a dyadic primitive relation of connectivity between two regions (Santos and Shanahan [2002]). Regions in RCC are considered to be non-empty regular closed subset of the topological space. In context of RCC, the RCC-8 is a relation algebra based on eight *basic relations* that are possible between any two regions. These basic relations of RCC-8 are *jointly exhaustive and pairwise disjoint (JEPD).*

In order to understand the basic relations of RCC-8 let us assume that variables `x` and `y` take on values from the domain of regions in $\mathbf{R}^2$. Each of these regions is composed of an interior and a boundary. Let x' and y' be the variables representing interiors of x and y respectively. For this domain following are the definitions of RCC-8 basic relations:

Definition 2.5.1: Relation `DC(x,y)` is true when `x` is *disconnected* from `y`.

$\forall_x \forall_y$ `DC(x,y)` $\iff$ `x` $\cap$ `y` $= \varnothing$

Definition 2.5.2: Relation `EC(x,y)` is true when `x` is *externally connected* with `y`.

$\forall_x \forall_y$ `EC(x,y)` $\iff$ `(x` $\cap$ `y` $\neq \varnothing$`)` $\wedge$ `(x'` $\cap$ `y'` $= \varnothing$`)`

Definition 2.5.3: Relation `TPP(x,y)` is true when `x` is a *tangential proper part* of `y`.

$\forall_x \forall_y$ `TPP(x,y)` $\iff$ `(x` $\subset$ `y)` $\wedge$ `(x` $\not\subseteq$ `y')`

Definition 2.5.4: Relation `NTPP(x,y)` is true when `x` is a *non tangential proper part* of `y`.

$\forall_x \forall_y$ `NTPP(x,y)` $\iff$ `(x` $\subset$ `y')`

Definition 2.5.5: Relation `PO(x,y)` is true when `x` is *partially over* `y`.

$\forall_x \forall_y$ `PO(x,y)` $\iff$ `(x'` $\cap$ `y'` $\neq \varnothing$`)` $\wedge$ `(x` $\not\subseteq$ `y)` $\wedge$ `(y` $\not\subseteq$ `x)`

Definition 2.5.6: Relation `EQ(x,y)` is true when `x` is *equal* to `y`.

$\forall_x \forall_y$ `EQ(x,y)` $\iff$ `x` $=$ `y`

Definition 2.5.7: Relation `TPPi(x,y)` is true when `y` is a *tangential proper part* of `x`.

$\forall_x \forall_y$ `TPPi(x,y)` $\iff$ `(y` $\subset$ `x)` $\wedge$ `(y` $\not\subseteq$ `x')`

Definition 2.5.8: Relation `NTPPi(x,y)` is true when `y` is a *non tangential proper part* of `x`.

$\forall_x \forall_y$ `NTPPi(x,y)` $\iff$ `(y` $\subset$ `x')`

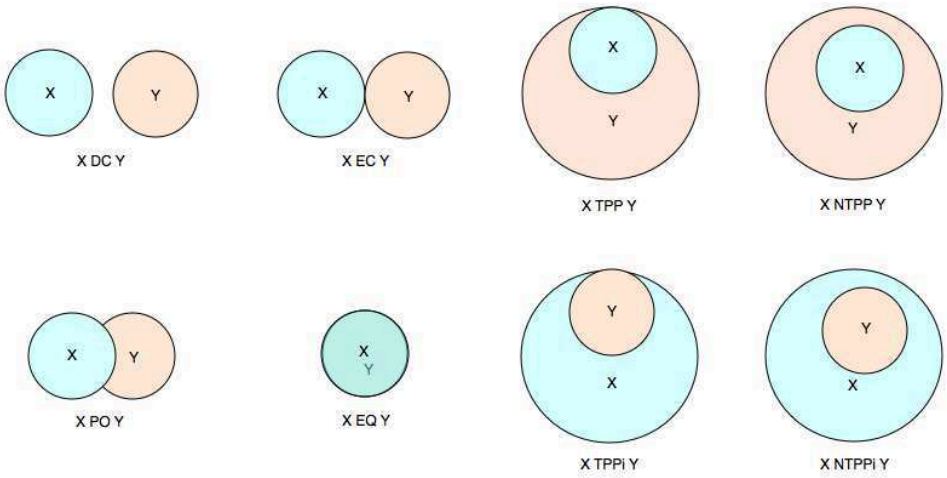


Figure 2.2: RCC-8 basic relations between planar regions X and Y.

Figure 2.2 gives a graphical representation of the basic RCC-8 relations. This work uses the above discussed relations to derive a vocabulary of basic topological concepts about space. This vocabulary also includes concepts regarding direction and motion of the objects. A detailed discussion of this vocabulary is given in chapter 6.

2.5 Fault diagnosis

A fault is an unpermitted deviation of at least one characteristic property or parameter of the system from the acceptable, usual or standard condition R. J. Patton [2000]. In order to diagnose a fault, a system first needs to *detect* its presence and the time of its occurrence. This phase is commonly known as *fault detection*. After the detection, the fault is *isolated* by determining its location and kind. This process is known as *fault isolation*. The fault isolation phase is a precursor to *fault identification* phase in a fault diagnosis system. The fault identification corresponds to knowing the behavior of the occurred fault.

In this work we consider a robot and its environment to be our system. In such cases we can categorize the faults into two broad categories (Akhtar [2011]).

1. Internal faults:
   These are the unpermitted deviations in the properties or parameters of the internal

components of a robot. These faults can be isolated to the internal components of the robot (e.g. robot's sensors or actuators).

2. External faults:
   These are the unpermitted deviations in the properties or parameters of objects that are present in the environment of a robot. We consider such objects to be the part of our system because in this work we are interested in the robots which interact with the objects to achieve their goals (i.e. mobile manipulators). Although detection of the external faults can be made by the (software) components of a robot, however they can not be isolated to the robot's internal components. This happens because of the fact that occurrences of these faults do not take place in the internal components of a robot. Therefore, external faults can only be located in the environment of the robot[5]. Examples of external faults in robotics can be found in chapter 4.

In this work we refer to the *external faults* as *unknown external fault*. We do this because the term *unknown external faults* emphasizes the fact that reasons behind the occurrences of these faults are the unforeseen situations which remain *unknown* until their occurrences.

[5]It is assumed that no internal fault occurs when the external fault is detected.

3 State of the Art

It is an obvious fact that reliability against faults is considered an important issue in robotic systems. R. J. Patton [2000] considers fault tolerance and diagnosis as one of the major challenges for robotics and AI community. In robotics, dealing with faults is an active research area because, as mentioned by Carlson and Murphy [2003], it is a common experience in robotics that even the most carefully designed robotic systems encounter faults. Diagnosis of these faults typically requires tracking a very large number of possible faults in a complex non-linear dynamic systems with noisy sensors (Verma and Simmons [2006]). This fact makes model based diagnosis a common approach in dealing with robotic faults. A typical example of model based approach in robotic faults is Honghai Liu [2005], where the authors develop a model based approach called 'first priority diagnostic engine'. This approach detects the faults in the internal components of a system by continuously monitoring the parameters of the effectors of the system and based on the fault detection information it narrows the possibilities of the faulty components of the system.

When it comes to modeling of a *system* it is important to define its boundaries. In most of the works related to robotic faults, a robot itself or a sub part of a robot is considered as a system. This implies that these works deal with the robotic faults which are caused by failures or malfunctioning of *internal* components of the robots. For example, Verma et al. [2004] is mainly concerned with the faults caused by mechanical component failures (e.g. broken motors or gears), sensor failures (e.g. broken encoders) and malfunctioning of wheels etc. Similarly, the approach presented in Monteriu et al. [2009] is concerned with faults in the sensors of a robot. Monteriu et al. [2009] presents a model based fault detection and isolation system applied to sensors of an unmanned ground vehicle. In the works concerned with the internal faults of robots, there are also approaches which pay special attention to on-board computational capabilities of robots. For instance, Verma and Simmons [2006] proposes an approach that takes advantage of the structural information in the domain of the components of a robot to dynamically concentrate the computation to the most probable area without losing track of less likely areas. Some works in robotics (e.g. Pettersson et al. [2007]) also prefer model-free approaches for dealing with faults. One of the major motivation behind model-free approaches is the fact that model based approaches do not show graceful degradation[1].

In fault diagnosis approaches many researchers also exploit qualitative reasoning. For instance, de Kleer and Williams [1987] uses qualitative reasoning to diagnose faults in

[1] Simmon R., Fernandez J., Golden K., Joskowicz L., Pollack M., Model Based Monitoring and Diagnosis for Mobile Robots. http://www.cs.cmu.edu/rll/overview/reids02.

composite devices. This is done by predicting behaviors of the devices by qualitatively inferring about them with the help of the information on the structure and components of the devices. Using qualitative reasoning in fault diagnosis provides computational advantages and it is considered very effective in general. However, because of the limitations of the process of qualitative abstraction, qualitative reasoning in fault diagnosis is mainly restricted to composite devices and well behaved systems. Examples of works in this regard can be found in Weld and Kleer [1990]. There are also few works related to robotics that take advantage of qualitative reasoning in dealing with faults. For instance, Honghai and Coghill [2005] represents an approach called 'unit circle' that models kinematics of a robot qualitatively and uses this model to deal with faults in the internal components of a robot. Daigle [2008] also presents an approach for event-based diagnosis of hybrid systems. This is a model based approach that uses qualitative abstraction of deviations in the behavior of the system from their nominal values. In our previous work (Akhtar [2011]) we also use qualitative reasoning in dealing with *unknown external faults* in robotics. We use qualitative version of physical laws on naive physics concepts to reason about unknown external faults.

Although most of the literature about fault diagnosis in robotics deals with internal faults, however examples of occurrences of unknown external faults can be found in robotics literature. For instance, Okada et al. [2008a] reports occurrence of an unknown external fault in the form of slipping of a bottle from the hand of HRP2JSK humanoid robot. It should be noticed that the group which reports this fault is particularly interested in robustness in the actions of the robot. It is claimed by this group that they have been able to achieve high robustness in the behavior (i.e. different actions) of the robot with the help of reliable hardware and careful software development. However, slipping of objects from the robot's hand still causes the robot to sometimes fail an action. Reports of unknown external faults can be witnessed in robotics literature under words like 'interaction faults', 'unforeseen situations' , 'unexpected events' and '(unfavorable) environmental conditions' etc. For instance, Steinbauer [2011] uses such terms in discussing the robotic faults which can be categorized as unknown external faults.

Currently, researchers in robotics are also interested in the external faults that are caused by unforeseen events in the environment of a robot because of presence of external agents (e.g. human beings). In this regard, Karg et al. [2011] proposes to formalize the understanding of 'normality' of behaviors of human beings which can be used in detecting and handling external faults. Another example of dealing with external faults in robotics is Ueda et al. [2011]. In this work the authors are concerned with re-planning of robotic tasks in case of occurrence of unknown external faults in presence of other agents. The authors present an architecture of a system that uses sensory feedback to re-plan the action of a robot when it faces occurrence of external faults.

4 On the nature of unknown external faults

In this chapter we give insights about the nature of *unknown external faults*. We highlight few important observations regarding these faults in the context of manipulation tasks in robotics. These observations provide the rationale for the approach developed in this work for increasing reliability of mobile manipulators against such faults.

(a) Situation 1 (b) Situation 2 (c) Situation 3

Figure 4.1: Situations in a robot's environment.

Consider the situations shown in figure 4.1. In situation '1', a NAO[1] robot is performing a task of placing a die on a cube. In this task the die falls on the floor instead of staying on the cube after the robot releases the die. The main reason for the unsuccessful completion of the robot's task is that the robot releases the die in an incorrect orientation, because of which it falls on its edge on the cube and finally falls on the floor. In figure 4.1(b), a Care-O-bot 3[2] drops an object into a basket. This task is completed successfully. However, consider a simple variation in the current situation in which the basket already contains some other object(s). In such a case, it is possible that the dropped object hits the other object(s) and finally falls outside the basket. Similarly, for the situation shown in figure 4.1(c) if the robot releases the object in the current pose of the object then it is possible that the object falls on its side instead of standing straight. This can eventually cause the object to fall on the floor instead of staying on the table.

In the situations discussed above and many other similar situations, a robot is likely to fail in completing its task successfully despite perfect functioning of its sensors and actuators. This happens because the causes of such failures reside in a robot's environment rather than in its internal components. Additionally, the anomalous situations which result in the failures are usually unforeseeable at the development phase of a robotic system and

[1] http://www.alderbaran-robotics.com
[2] http://www.care-o-bot-research.org

they remain unknown until their occurrence. For a robot, occurrence of such situations is exhibited as *unknown external faults*, which prohibit the robot from completing its tasks successfully. The *unknown external faults* manifest themselves as deviations in the values of certain properties of objects (in the robot's environment) from their expected or predicted values. We can make following useful observations related to *unknown external faults* by examining the above mentioned situations:

1. *Situations exhibited as unknown external faults are governed by physical laws.* In other words, knowledge of physical laws can not only provide insights in reasoning about fault occurrence but it can also provide means to avoid the situations that result in the faults.

2. *It is possible to detect unknown external faults by noticing mismatches in the predicted and the observed behavior of the manipulated object.* For the case of plan-based robotic systems the mismatches can be detected by the unsatisfied *effects* of the actions performed by the robot.

3. *Cause of an unknown external fault can be isolated to the action who's effects remain unsatisfied* after the action has been completed. It can be argued that some external faults only manifest themselves in later actions while their causes actually occur in some previous action. For instance, the robot in figure 4.1(a) releases the die in an incorrect orientation because it picks up the die in an incorrect manner. Although this argument is valid, however we argue in favor of associating a fault only to the action with unsatisfied *effects*. We do this because of following reasons:

   - Not every anomaly in some previous action results in a fault in some future action. For example, in the situation of figure 4.1(a), if the robot releases the die over the exact center of the cube the task may still be completed successfully (despite the incorrect picking of the die).

   - Any detected anomaly in some previous action can be rectified in later actions before the actual occurrence of the fault. This rectification can be postponed as late as the last action before the occurrence of the fault.

   - Considering the above statement, it is always possible to modify the action that causes the fault detection, in a way that the fault can be avoided in the future.

   - Monitoring and modifying only a single action is both simpler and computationally effective as compared to tracking the anomalies in all the previously performed actions and making modifications based on such observations.

4. *Situations which result in a fault occurrence only involve those actions of a robot in which it physically interacts with the objects.* It is worth noticing here that we limit

the scope of the sources of *unknown external faults* to natural physical phenomena only. We do not consider the cases in which an external agent (e.g. other robots, human beings) is the cause of a detected fault.

5. *Occurrence of the faults depends on the intrinsic properties of the objects.* In figure 4.1(a) and (c) it is clear that the situations can vary greatly by changing the intrinsic properties (e.g. shape, size) of the manipulated objects. Therefore, to improve a robot's reliability against *unknown external faults* any approach must take care of low level details regarding the properties of the objects.

6. *There is no absolute a priori assurance against unknown external faults.* By the very concept, an *unknown external fault* represents a situation that has not been taken care of in the development phase of a robotic system, nor it has been modeled in the robot's diagnostic system. Once an *unknown external fault* has been detected and some measure has been taken by the robot to avoid its occurrence again, it is possible that the robot fails the same action again because of some other unseen situation.

   Although, it may seem at this point that taking care of an (already occurred) fault is futile, however this is not true. For instance, consider the robot in figure 4.1(a). Assume that after the occurrence of the fault, the robot is somehow able to understand that it can perform the action more reliably if it releases the die over the center of the cube in a better orientation. If the robot performs the same action again it is more likely to perform it successfully than earlier. This is because, for this action any other unforeseeable situation resulting in an *unknown external fault* will most likely be less frequently occurring than the one that has already been taken care of. In other words the *mean time between failure*[3] (MTBF) of the action will increase.

7. *The best way to increase reliability against unknown external faults is to enable the robot to estimate the best way of performing the action* instead of separately taking care of each occurring fault. In a given situation if a robot is able to estimate the best (and the worst) way(s) of performing an action by itself, then it can avoid the occurrence of the unforeseen situations which could result in *unknown external faults.* Such an ability of the robot can make it reliable against the *unknown external faults* in the most effective manner.

The above given observations regarding the *unknown external faults* form the rationale of the approach proposed in this work. Although in the chapters to follow we do not refer to these observations very often, however insights from these observations are at the very crux of this work.

[3]See appendix (A) for the definition.

5 Approach

5.1 Problem specification

Once again, consider the NAO robot shown in the figure on the right. This `robot` performs a task with a goal of placing a `die` on a `table` (i.e. the cube). In order to complete its task the `robot` performs a sequence of actions that corresponds to picking up the `die`, walking towards the `table` and then putting the `die` on the `table`. For a plan based robotic system, the action of putting the `die` on the `table` is performed after instantiation of a planning operator similar to the one shown in figure 2.1. Below, we repeat the definition of the planning operator by instantiating it with the relevant objects from the situation shown in the figure. Now, this definition corresponds to the action of putting the `die` on the `table`.

```
Put-on(Die, Table).
preconditions: Holding(Robot, Die), ¬On(Die, Table), Empty(Table).
effects:       On(Die, Table), ¬Holding(Robot, Die).
```

As mentioned in chapter 4, the action shown in the figure above is not completed successfully by the `robot` and the `die` falls on the floor instead of staying on the `table`. This fact is noticed by the robot when the predicate `On/2` in the *effects* of the performed action is not satisfied by the state of the objects observed after the completion of the action. In our settings, this causes the *detection* of an *unknown external fault*, which can be immediately *isolated* to the action of `Put-on`. It is noticeable that here the *detection* and the *isolation* of the fault is based on the model of the behavior of the system which is already known to the robot in the form of a planning operator. The process of *detection* is carried out by comparing the predictions (i.e. the *effects*) of the outcome of the performed action with the observed outcome.

In the above illustration of occurrence and detection of an *unknown external fault* we make few assumptions about the `robot` and the situation, which are also valid for the approach presented in this chapter. Following are these assumptions:

- The fault is indeed an *unknown external fault* and no internal component failure or malfunctioning is detected by the robot.
- The model of the behavior of the system[1] (i.e. the planning operators) is already available to the robot.
- The robot is able to correctly evaluate the predicates in the *effects* of the actions by continuously monitoring the behavior of the system.
- There is no involvement of external agents in the occurrence of the faults.

We know from section 2.3 that a plan based robotic system can only execute an action when the current state of the system satisfies the *preconditions* of that action. This means, in our running example following three conditions are true in the state before the robot actually executes the action of `Put-on` that results in the detection of an *unknown external fault*:

- The `robot` is *holding* the `die`. (i.e. `Holding/2` predicate is true.)
- The `die` is *not on* the `table`. (i.e. `On/2` predicate is false.)
- The top surface of the `table` is *empty*. (i.e. `Empty/1` predicate is true.)

Despite the fact that our system is in the perfect state (according to the *preconditions* of the action), the system encounters a fault. From this, we can conclude that the occurrence of the fault is because of the fact that the *preconditions* of the executed action put insufficient constraints on the state of the system. In other words, the *preconditions* of the actions which result in the fault need to consider more details regarding the state of the world. For instance, in our example the *preconditions* of `Put-on` do not consider the fact that if the `die` is released by the `robot` over a `table` (with a smaller top surface), then the orientation of the `die` should also be kept correct. If the robot in the figure above, considers this fact and releases the `die` in the correct orientation then it can complete its action successfully.

It is true in general that occurrence of *unknown external faults* can be mitigated by considering more details about the state of the system in the *preconditions* of the actions which cause the detection of the fault. In this work we propose an approach that mitigates the occurrence of *unknown external faults* by incorporating more details, about the object(s) manipulated by the robot, in the *preconditions* of the actions that result in faults. These details are included in the *preconditions* without disturbing the definitions of the predicates already present in the *preconditions*. This is done by including another predicate `Allowed/N` in the *preconditions* of the planning operator of the action that results in an *unknown external fault*. The predicate `Allowed/N` evaluates to true only when the current

[1]The word *system* corresponds to the robot and the objects involved in the performed action.

state of the objects involved in the action satisfies the constraints that are necessary to perform the action successfully. This enables the robot to perform its actions more reliably by avoiding the occurrences of *unknown external faults*.

The planning operator in figure 5.1 shows the modification to the definition of `Put-on` after inclusion of the `Allowed/N` predicate, according to the proposed approach. In this definition `N` = 3. The first element in the argument of `Allowed/3` is the name of the planning operator and rest `N-1` arguments are the objects involved in the definition of the planning operator. The name of the planning operator is a constant symbol, whereas other arguments are variables. Normally, the definitions of planning operators do not include constant symbols. We use the constant symbol only to emphasize the correspondence between the operator and the new predicate.

```
Put-on(object-1, object-2).
preconditions: Holding(agent, object-1), ¬On(object-1, object-2),
               Empty(object-2), Allowed(Put-on, object-1, object-2).
effects:       On(object-1, object-2), ¬Holding(agent, object-1).
```

Figure 5.1: Modified planning operator.

In this work we focus only on those actions of a manipulator which involve the action of releasing of an object. Since the target application of this work is assumed to be house hold mobile manipulators, therefore the objects considered in this work are mainly house house hold items. It is assumed in this work, that the robot that manipulates the objects is able to detect and isolate the *unknown external faults* as explained above. The robot triggers the approach presented below after the detection of a fault, to perform the action more reliably in the future. Although the scope of this work is limited to the actions which involve simple release of objects over other objects, but the explanation of the proposed approach uses a more general language. This is done because it is believed that the formalization of the problem; the concepts discussed in the approach; and the methods and algorithms presented in the approach can also be applied[2] to other similar actions and their scope is much broader.

5.2 Schematics

Figure 5.2 shows the basic schematics of the approach developed in this work for increasing reliability of robotic manipulation tasks against *unknown external faults*. The approach primarily consists of three steps (shown as 1, 2 and 3) which modify a planning

[2]With necessary changes.

operator in a manner discussed in section 5.1. Step 0 is included in the figure to illustrate the inputs expected by the approach. A detailed explanation of each step is given below.

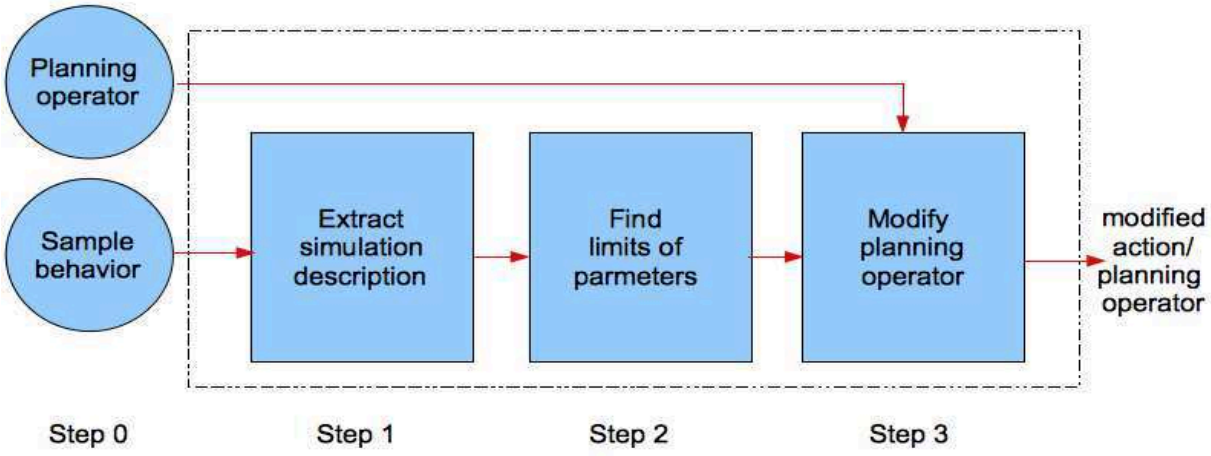


Figure 5.2: Basic schematics of the approach.

5.2.1 Step 0

The proposed approach expects two inputs:

1. The definition of the planning operator that is involved in the detection of the *unknown external fault*.
2. A sample behavior of the relevant objects that can be simulated to show a successful execution of the (concerned) action.

Since we assume that the planning operators conform to classical representation, the definition of the planning operator is expected to be similar to the one shown in figure 2.1. On the other hand, the sample behavior represents a construct that comprises:

- Δ: A *list* of parameters of the manipulated object which are required to simulate the desired behavior of the objects.
- $\boldsymbol{\mu}$: A *vector* consisting of the values of the parameters in Δ.
- CAD *models* of the objects involved in the action.

With the help of the sample behavior it should be possible to simulate the desired behavior of the objects that are involved in the action. In this work we use Open Robotics Automation Virtual Environment (OpenRAVE) (Diankov [2010]) for the simulation of objects' behaviors, which in turn uses Open Dynamic Engine (ODE) (Smith [2007]) for simulating rigid body dynamics. Table 5.1 shows the parameters in Δ that are used in

the experiments performed (in chapter 7) with the proposed approach. The symbols of the parameters show the notations with which the parameters are henceforth denoted. The parameters in Δ are properties of the object that is manipulated by the robot in an action. The reader is reminded that previously we have referred to the manipulated object as `object-1`. Coming sections/chapters of this work also observe this convention. At this point, it should also be noticed that in this work we consider the actions which involve only two objects. Hence, the sample behavior includes the models of both the objects (i.e. `object-1` and `object-2`).

Parameter	**Symbol**
x-coordinate	x
y-coordinate	y
z-coordinate	z
Roll	ρ
Yaw	θ
Pitch	ϕ
x-component of linear velocity	$\dot{x}$
y-component of linear velocity	$\dot{y}$
z-component of linear velocity	$\dot{z}$
x-component of angular velocity	$\dot{\rho}$
y-component of angular velocity	$\dot{\theta}$
z-component of angular velocity	$\dot{\phi}$

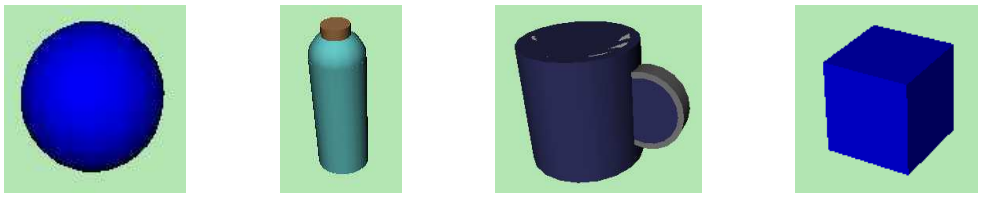
Table 5.1: Parameters of Δ and their symbols.

Occurrence of an *unknown external fault* can be described as a problem in which a `robot` expects `object-1` to be in a particular *goal state* after releasing it in an *initial state*, but the object ends up in some *final state* that is not the same as the expected *goal state* (Akhtar [2011]). For instance, in the example shown in section 5.1, the `die` is released by the `robot` in an *initial state* that is visible in the figure. The `robot` expects it to stay on the `table` (i.e. its *goal state*), but the `die` ends up in a *final state* of being on the floor. Above formalization of the problem of *unknown external faults* is valid in general for the manipulation tasks which involve releasing of an object. The approach presented in this work also takes advantage of the same formalization. Further details in regard of formalization of the problem can be found in Akhtar [2011] and Akhtar and Kuestenmacher [2011].

The values of the parameters in Δ that are provided in the vector $\boldsymbol{\mu}$ correspond to the values that represent the *initial state* of `object-1`. It is mandatory for the proposed approach that the *initial state* represented by $\boldsymbol{\mu}$ results in the *goal state* of `object-1`. This can be guaranteed by simulating the objects and observing the behavior of `object-1` by setting its *initial state* to the desired values. In other words, the *sample behavior* in figure 5.2 can be seen as a working simulation of the desired/expected behavior of the

objects involved in the simulation. This behavior is the one that is expected from the objects if the action (defined by the planning operator) is carried out by the `robot` in a successful manner. Since we are interested in avoiding the *unknown external faults* by putting constraints only on `object-1`, we do not consider presence of the `robot` in the simulation.

For practical purposes, we limit the domain of `object-1` only to the solid objects which can be modeled as a combination of cube(s), cylinder(s) and sphere(s). Examples of such models are shown in figure 5.3.



(a) A ball (b) A bottle (c) A mug (d) A die

Figure 5.3: Examples of models for `object-1`.

We also limit the domain of `object-2` to the solid objects which have the geometric shapes of their top surfaces[3] as those shown in figure 5.4. All of these shapes represent convex regular shapes. We exploit regularities of these shapes and those of `object-1` models only in simplifying the geometric calculations in the implementation. Otherwise, the developed approach does not take advantage of the regularities of the geometries of the objects. This has been done intentionally and (in principle) it makes the proposed approach also valid for other convex shapes for `object-2` and other models for `object-1`. This issue will be highlighted further in the coming sections of this chapter.

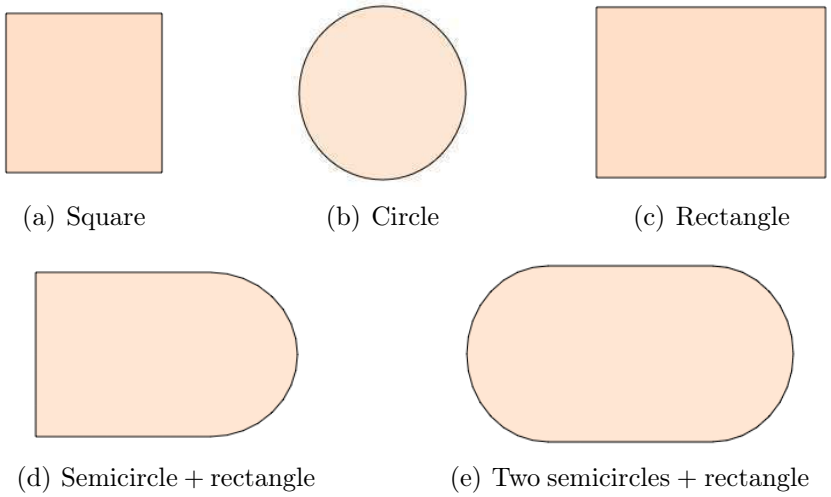
5.2.2 Step 1

As shown in figure 5.2, *step 1* requires only the sample behavior of the objects as its input. In this sample behavior, the information provided by $\Delta, \boldsymbol{\mu}$ and the models of the objects can be utilized to simulate the desired behavior of the objects. Furthermore, it is also possible to find a symbolic description of the simulated behavior. This step of the approach finds this description. This description is composed of conjunctions of predicates that are true at particular instants in the simulated behavior. For instance, in our running example from section 5.1, the objects can have following description at the instant of the release (i.e. the *initial state*) of the `die` in the simulation.

`Over(Die, Table)` $\wedge$ `ArbitraryOrientation`[4]`(Die)`.

[3] In chapter 7, we also experiment with a model of a basket. In that case the top surface implies the top surface of the bottom of the basket.

[4] `ArbitraryOrientation/1` is true when the orientation of the object in its argument is neither parallel to any axes nor to any plane formed by any two of the axes of the concerned frame of reference.



(a) Square (b) Circle (c) Rectangle

(d) Semicircle + rectangle (e) Two semicircles + rectangle

Figure 5.4: Geometric shapes for the top surfaces of `object-2`.

The above sentence in FOL describes only the instant of the *initial state* of the `die`. In our approach, we characterize the behavior of the objects by the instants representing the *initial* and the *final states* of the objects. Therefore, the description of the simulation (i.e. the desired behavior of the objects) is composed of two FOL sentences similar to the one shown above. Although at this point it may seem redundant to find a description of objects' behavior which can already be simulated, however upcoming sections of this thesis show that this is not the case. We defer the discussion on the benefits of *step 1* to later sections and focus only on technical details in this section.

The process of extracting the description of the simulation can be summarized as following sequence of steps:

1. Place *markers* on the objects.
2. Find description of the *initial state*.
3. Simulate.
4. Find description of the final state.

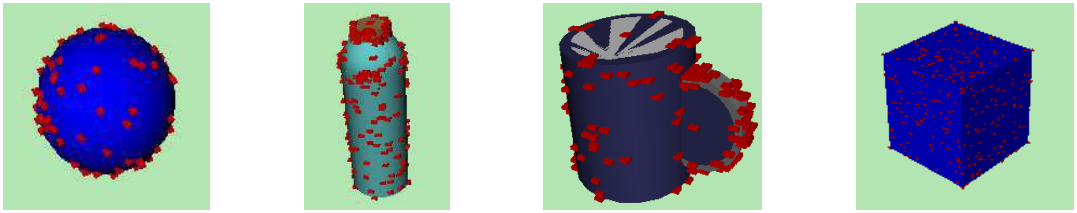
Figure 5.5: Summary of the process of extracting description of simulation.

We place *markers* on the objects in order to ease the process of finding the description of the states. For `object-1`, a marker $\mathcal{M}_{obj1} = [\mathbf{v}_{obj1}, s_{obj1}]$, where:

- $\mathbf{v}_{obj1}$ is a vector composed of x, y and z coordinates of a point on the surface of `object-1`. The coordinates are measured in the frame of reference attached to `object-1`.

- s_{obj1} is a logical symbol (i.e. a constant) such that $s_{obj1} \in A_1$, where $A_1 = \{$ *top, bottom, right, left, front, back, none*$\}$.

The coordinates in a vector $\mathbf{v}_{obj1}$ represent a random point on the surface of `object-1`. We use uniform random distribution to generate such points. Figure 5.6 shows the models of the objects with the vectors of the markers shown as red points[5]. The symbols s_{obj1} attach semantics to *each* marker. In the elements of A_1 each symbol represents exactly the same meaning as suggested by its name. For example, the symbol *top* corresponds to a marker that has $\mathbf{v}_{obj1}$ with maximum z value. It is possible that there are more than one markers that satisfy the definition of a single symbol. In that case we break the tie by randomly selecting one marker. With the exception of the symbol *none*, we associate each element of A_1 with exactly one marker.



(a) A ball (b) A bottle (c) A mug (d) A die

Figure 5.6: Examples of models for `object-1` with markers.

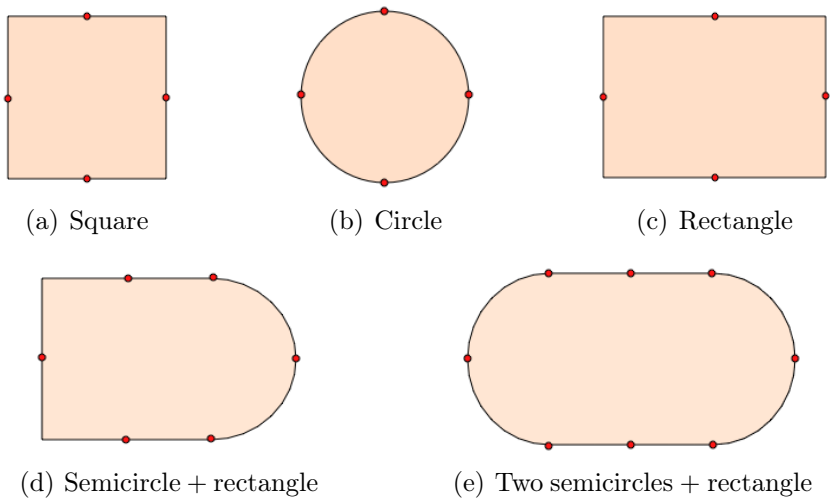
For the case of `object-2` a marker $\mathcal{M}_{obj2} = [\mathbf{v}_{obj2}, s_{obj2}, g_{obj2}]$, where:

- $\mathbf{v}_{obj2}$ is a vector composed of x, y and z coordinates of a point on the top surface of `object-2`. The coordinates are measured with respect to a frame of reference attached to `object-2`.
- s_{obj2} is a logical symbol such that $s_{obj2} \in A_2$, where $A_2 = \{$ *right, left, front, back, none*$\}$.
- g_{obj2} is a list composed of a symbol for the shape of the geometries (e.g. *rectangle, circle*) of the top surface of `object-2` and a vector for the location of the center points of these geometries, measured in the reference frame attached to `object-2`.

For `object-2`, a vector $\mathbf{v}_{obj2}$ represents a point that is located on the boundary of the top surface of the object. Figure 5.7 shows the example of the points on the top surfaces of the objects considered in this work. It can be seen that unlike the case of `object-1`, these points are chosen to be at specific locations. In the figure 5.7(a), (b) and (c), the points are exactly at the locations which represent concepts denoted by the first four elements of A_2. Whereas, figure 5.7(d) and (e) also include points that correspond to

[5]These points do not physically exist in the models. Points shown in figure 5.6 and later in figure 5.7 are only to illustrate the concept.

the *none* symbol. These points are also specifically chosen to be at the locations of coincidence of the boundaries of the semi-circles and the rectangles. We choose these specific locations for the points to simplify geometric calculations in the implementation of the approach.



(a) Square (b) Circle (c) Rectangle

(d) Semicircle + rectangle (e) Two semicircles + rectangle

Figure 5.7: Geometric shapes for the top surfaces of `object-2` with markers.

In addition to simplification, we choose the geometries (and hence the locations of the markers) shown in figure 5.7 for the top surfaces of `object-2` because these geometries already represent the top surfaces of a large group of objects (e.g. tables, stools) which are found in a daily life environment. Since we are interested in the robotic tasks that involve the release of an object over another object, the simplified implementation already covers most of the cases for `object-2` in house hold or service robotics environments. In case the approach is required to be applied to other objects (for `object-2`) with convex top surfaces, following two issues must be taken care of:

1. For each marker $\mathcal{M}_{obj2}$, elements of A_2 must be associated with $\mathbf{v}_{obj2}$ just like the case of `object-1`. This also means that there will be no need for g_{obj2} in the definition of a marker for `object-2`. This happens because we use the information provided by g_{obj2} only for the simplification of geometric calculations for the shapes shown in figure 5.7.

2. The implementation must take care of the approximations of the shapes of the top surfaces of `object-2`. In our work, one of the functions of the markers shown in figure 5.7 is to approximate the boundaries of the shapes on which the markers are placed. As will be discussed in chapter 6, we exploit this approximation in evaluating the predicates used for logical description of the behavior of the objects. In case of regular geometries with the configuration of the markers shown in figure 5.7, it is possible to know about the exact boundaries of the objects. It is possible that for other convex surfaces this information is not retrievable from the markers only. In

that case the implementation should take care of this issue either by approximating the boundaries of the shapes with the help of (more number of) markers or by some other suitable method. It is true that in case of approximation there will be some loss of information about the geometry of the shape. However, as will be apparent from the later parts of the thesis, minor loss of information will not have profound effects on the approach.

We use the semantics attached to the objects with the help of markers in evaluating the truth values of a group of predicates. This is the process of finding the description of the simulation of the objects. Since, we need a description at the *initial state* and at the *final state*, so the evaluation process is carried out twice. The transition between the *initial* and the *final* state is carried out by the simulation process which applies physics laws to the objects with the help of the physics engine. For our approach, the *final state* of the objects corresponds to the state when `object-1` becomes stationary in the simulation. This is owing to observation (1) in chapter 4 and the assumption that there is no involvement of any external agent during the occurrence of the fault.

In order to find the description of each state we use a group of predicates that are evaluated at the instants of the *initial* and the *final state* of the objects. These predicates have been systematically chosen and defined. We will focus on these predicates and their definitions in chapter 6, where we refer to these predicates collectively as *description vocabulary*. Here we want to highlight the point that because of high expressive power of FOL, it is possible to describe almost any state of the objects in an environment with the help of predicates. However, selection of suitable predicates with useful definitions is a major issue in this regard. The predicates used in this work are particularly suitable for describing the states of the objects in the tasks of robotics manipulation which involve release of an object over another object.

5.2.3 Step 2

After finding the description of the simulation, we find the limits of the values of the parameters in Δ. These limits correspond to the extreme values of the parameters which can be used by the robot and the robot can still perform the action successfully. In order to illustrate the working and aim of *step 2*, we introduce another example here. This example is based on one of the experiments in chapter 7.

Assume that a robot detects an *unknown external fault* in the action of placing a `bottle` on a `cube`. This triggers our approach by providing it with the *sample behavior* and the *planning operator* of the action which results in the fault. Under normal circumstances, the sample behavior should result in a simulation with the initial state of the objects

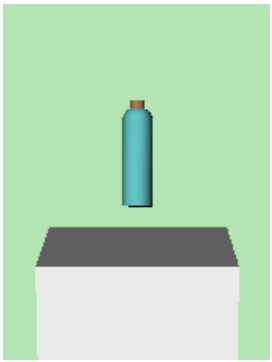


Figure 5.8: Initial state of releasing a bottle on a table.

similar to figure 5.8. On execution of the simulation this state transforms into a final state in which the `bottle` stands straight on the `cube`. If the initial state of the sample simulation is indeed similar to the one shown in figure 5.8, then *step 1* of our approach results in two FOL sentences $\mathtt{S}_{init}$ (for the *initial state*) and $\mathtt{S}_{final}$ (for the final state), where:

$$\begin{aligned}\mathtt{S}_{init} \equiv\ & \mathtt{Over(Bottle,\ Cube)} \wedge \mathtt{Zz(Bottle,\ Cube)} \wedge \mathtt{StraightAlong\text{-}z(Bottle)} \wedge \\ & \mathtt{Stationary\text{-}V(Bottle)} \wedge \mathtt{Stationary\text{-}AV(Bottle)}.\\ \mathtt{S}_{final} \equiv\ & \mathtt{On(Bottle,\ Cube)} \wedge \mathtt{Zz(Bottle,\ Cube)} \wedge \mathtt{StraightAlong\text{-}z(Bottle)} \wedge \\ & \mathtt{Stationary\text{-}V(Bottle)} \wedge \mathtt{Stationary\text{-}AV(Bottle)}.\end{aligned}$$

Each of the predicates in above sentences is true under certain conditions. We do not go into formal description of these conditions in this chapter of the thesis[6]. The issue to be noticed here is that both the sentences, $\mathtt{S}_{init}$ and $\mathtt{S}_{final}$ are conjunctions of five predicates. In our work this is true in general, therefore we can write a general form for the FOL expressions above as following:

$$\mathtt{S}_s \equiv \mathtt{P1} \wedge \mathtt{P2} \wedge \mathtt{P3} \wedge \mathtt{P4} \wedge \mathtt{P5}.$$

In this expression each predicate denotes a fact about a unique aspect of the objects involved in an action. For instance, `P3` (i.e. `StraightAlong-z/1` in our current example) states a fact about the aspect of *orientation* of the `bottle`. We also know that the *orientation* of an object can be described by the values of an object's *roll, yaw* and *pitch* parameters. Such an association between the parameters in Δ and the predicates in $\mathtt{S}_s$ exists for all five predicates. Table 5.2 shows this association for all five predicates.

Using the association between the parameters and the predicates we can also find the limits of the values of the parameters. For instance, in our current example the predicate `P3` (i.e. `StraightAlong-z/1`) states the fact that the bottle's orientation is straight along z-axis (of the frame of reference attached to `object-2`). This fact still remains true if we

[6]The conditions are formalized in terms of definitions in chapter 6.

Predicate	Aspect of objects	Parameters
P1	Connectedness between the objects	x, y, z
P2	object-1's direction relative to object-2	x, y
P3	Orientation of object-1	ρ, θ, ϕ
P4	Linear motion of object-1	$\dot{x}, \dot{y}, \dot{z}$
P5	Angular motion of object-1	$\dot{\rho}, \dot{\theta}, \dot{\phi}$

Table 5.2: Association of parameters with predicates.

rotate the bottle along the z-axis (of the said reference frame) by any angle. According to our convention this rotation represents ϕ (i.e. *pitch*) of the bottle. In other words, by knowing the P3 predicate we also know that, for $-\infty \leq \phi \geq \infty$ the state of the bottle (as described by $\mathtt{S}_s$) remains unchanged. Within this interval of values for ϕ we can choose the limits of ϕ to be $\frac{-\pi}{4} < \phi > \frac{\pi}{4}$. Limits of the other two parameters associated with P3 can also be found in a similar manner. However, for those limits we need to remind ourselves that we are interested in the limits of the parameters only in the *initial state* of the objects. Furthermore, the limits represent the extreme values of the parameters with which the task can still be performed in a manner that is similar to the *sample simulation*. Keeping in view these facts, we can see that it is possible to drop a bottle in a slightly tilted manner such that the bottle still stands straight on the cube in its *final state*. Therefore, we can have some non-zero upper and lower limits for ρ (i.e. *roll*) and θ (i.e. *yaw*) of the bottle. We choose these non-zero upper and lower limits to be $\frac{\pi}{16}$ and $\frac{-\pi}{16}$. That is, $\frac{-\pi}{16} \leq \rho \geq \frac{\pi}{16}$ and $\frac{-\pi}{16} \leq \theta \geq \frac{\pi}{16}$.

Let us denote the limits of each parameter δ (where, $\delta \in \Delta$), associated with a predicate Pn (where, $n \in$ [1,2,..,5]), by a closed limit interval $Lim_{Pn}\delta$. Except for $Lim_{P1}x$ and $Lim_{P1}y$, we can find values of all $Lim_{Pn}\delta$ in a manner similar to the one described in the previous paragraph. It is imperative to know about the formal definitions of predicates and geometric settings of these definitions to fully understand the reasons behind the values chosen for $Lim_{Pn}\delta$. Therefore, we leave the discussion on the exact values of the limits of the parameters for chapter 6. For the remaining description in this section it is enough to think of the values of $Lim_{Pn}\delta$ as an estimate of the extreme values of the parameters based on basic geometric and physics laws. These values provide the estimate of the limits of the parameters by considering complete independence between the parameters. Therefore, it is possible that for a particular combination of the values (chosen within the limits), the simulated action can not be completed successfully. However, this possibility does not impair the performance of our approach. In fact, for our approach it is desirable that within the values suggested by $Lim_{Pn}\delta$, there also occur combinations that cause the objects to show undesirable behavior. As will be seen shortly, these combinations serve the purpose of *negative examples* for the learning stage in our approach.

Assume that based on the description above, the approach is able to find the limits of each parameter δ for our example, as shown in the table 5.3[7]. The limits of x and y are based on $Lim_{P2}x$ and $Lim_{P2}y$ which are estimated with the help of the dimensions of the top surface of the `table`. These dimensions are 0.3(m) for both x and y. Henceforth, we refer to the limits of parameters estimated by the procedure discussed above, as the *initial limits* of the parameters.

Parameter(s)	**Lower limit**	**Upper limit**
x, y	-0.1875	0.1875
z	0.81	1.0
ρ, θ	$-\pi/16$	$\pi/16$
ϕ	$-\pi/4$	$\pi/4$
$\dot{x}, \dot{y}, \dot{z}$	-0.001	0.001
$\dot{\rho}, \dot{\theta}, \dot{\phi}$	-0.001	0.001

Table 5.3: Values of the initial limits of δ for example in figure 5.8.

In table 5.3, the initial limits of the parameters are based on $\boldsymbol{\mu}$ and the models of the objects in the *sample simulation*. Given $\mathtt{S}_{init}$ and $\mathtt{S}_{final}$, these limits can be estimated without simulating the objects. If the objects are simulated by randomly selecting values of the parameters from these limits, the *new* FOL expressions for the initial state (i.e. $\mathtt{NewS}_{init}$) and for the final state (i.e. $\mathtt{NewS}_{final}$) of the objects may change because of new predicate for `P1` (i.e. $\mathtt{P1}_{new}$). For instance, in our current example if we choose values of x and y to be -0.18 each, the bottle will no longer be `over` the table in its initial state and it will not fall `on` the `cube` when the simulation is executed. Hence, `Over/2` and `On/2` predicates in $\mathtt{S}_{init}$ and $\mathtt{S}_{final}$ will be replaced by some other predicates. For a simulation, it is possible to always keep $\mathtt{NewS}_{init}$ same as $\mathtt{S}_{init}$. This can be done by allowing the simulation to run for only those combinations of the values of parameters for which $\mathtt{P1}_{new}$ is same as `P1`. We denote any vector composed of such values of the parameters as $\boldsymbol{\mu}_{new}$.

If we neglect the issue of time, then it is possible to generate as many $\boldsymbol{\mu}_{new}$ vectors as possible. If we execute the simulation for each of $\boldsymbol{\mu}_{new}$, the final state of the objects may be the desired one for some vectors and undesired one for the others. We can label a desired $finalState$ as '1' and an undesired one as '0'. By doing so, we are actually generating labeled training examples which can be used as an input to a classification algorithm. In these examples an input vector consists of the values of $\boldsymbol{\mu}_{new}$ and the corresponding output label is the label of the $finalState$. With the help of the experience given as training examples, a classification algorithm can learn to predict the desirability of an initial state of the objects. For this work, figure 5.9 shows the process of generating the training examples for a classification algorithm in an algorithmic form. The (algorithm in

[7]Values of the limits are taken from actual experiment.

the) figure also shows that how the initial limits of the parameters can be further refined to new values such that these values also incorporate $Lim_{P1}x$ and $Lim_{P1}y$. Below we give an illustrative explanation of this algorithm.

1. let $limitsChanged = false$
2. *while* $simCount \leq desiredExamples$:
3. *if* $limitsChanged = false$ and $simCount > 10\%$ of $desiredExamples$ and $min(posExpCount, negExpCount) > 1\%$ of $desiredExamples$:
4. *then* change the limits of each δ such that the lower limit represents the smallest value of δ generated so far and the upper limit represents the largest value generated so far for δ and
5. let $limitsChanged = true$
6. *while* `true`:
7. *for each* δ, select values from a uniform random distribution defined by the current limits of the parameter.
8. Evaluate $\texttt{NewS}_{init}$ using the new values δs.
9. *if* $\texttt{P1}_{new} \equiv \texttt{P1}$ in $\texttt{NewS}_{init}$:
10. *then* store the values of δs as a vector $\boldsymbol{\mu}_{new}$ and
11. break.
12. *else* count the evaluation of $\texttt{NewS}_{init}$ as an $extraEvaluation$.
13. Simulate, using $\boldsymbol{\mu}_{new}$.
14. Evaluate $\texttt{NewS}_{final}$ using the values of parameters at objects' *final state*.
15. *if* $\texttt{P1}_{new} \equiv \texttt{P1}$ and $\texttt{P3}_{new} \equiv \texttt{P3}$, in $\texttt{NewS}_{final}$:
16. *then* $finalState = 1$ and increment $posExpCount$.
17. *else* $finalState = 0$ and increment $negExpCount$.
18. Append $finalState$ at the end of $\boldsymbol{\mu}_{new}$ to create a *training example*.
19. Increment $simCount$.

Figure 5.9: Algorithm for generating final limits of the parameters and generating training examples for a binary classification algorithm.

Assume that in our running example we need to refine the initial limits of the parameters (shown in table 5.3) and we also want to generate 5000 training examples for the action of placing the `bottle` on the `cube`. For this purpose the algorithm in figure 5.9 runs until the simulation count (i.e. $simCount$) becomes 5000 (i.e. number of $desiredExamples$). In each run, the simulation is executed by selecting values of each parameter for the initial state of the object from a uniform random distribution that is defined by the initial limits of that parameter. Before the execution of each simulation, the algorithm makes sure that following two conditions are true:

1. For the chosen values of the parameters, `P1` does not change its value in the *initial state* of the objects (line 6 to 11 in figure 5.9).

2. If a minimum number of training examples have been generated, then the initial limits are updated based on the values of the parameters generated so far (line 3 to 5 in figure 5.9).

Condition (1) should be true for a training example because the initial limits do not consider the limits placed on x and y parameters because of `P1`. That is, $Lim_{P1}x$ and $Lim_{P1}y$ are unknown at the start of the algorithm in figure 5.9. Therefore, random values of x and y parameters should only be considered valid for a simulation if they conform to `P1`. Condition (2) is about updating the values of initial limits of the parameters *once* in the complete execution of the algorithm. This update is made when the number of positive examples (i.e. $posExpCount$) as well as the number of negative examples (i.e. $negExpCount$) generated in the simulation process are larger than 1% of the number of $desiredExamples$. Furthermore, to make the update it is also made necessary that $simCount$ is already more than 10% of the $desiredExamples$. These conditions make sure that we have enough samples of both positive and negative examples to update the initial limits on the parameters. The process of update simply changes the limits of each parameter to the extreme valid values which fulfill condition (1). In each simulation, once a simulation is run using the values in $\boldsymbol{\mu}_{new}$, the objects transit to the final state. That is, in our example the `bottle` falls and attains a stable position on the `cube`. We consider the behavior of the `bottle` to be desirable if in its final state, the `bottle` is `on` the `cube` and it stands `straight along z` axis of the reference frame attached to the `cube`. In other words, we can label the $finalState$ as 1 if the value of `P1` and `P3` in the new simulation (i.e. $\texttt{P1}_{new}$ and $\texttt{P3}_{new}$) are same as the values of `P1` and `P3` in the sample simulation. Otherwise, we can label the $finalState$ as 0. The algorithm attaches this label to the $\boldsymbol{\mu}_{new}$ vector to generate a training example.

If we apply the algorithm in figure 5.9 to our running example with objects models shown in figure 5.8, the values of limits of x and y in the table 5.3 change. Now, -0.122 and 0.122 become the new lower and upper limits respectively for both x and y. Although it appears that the process of refining the limits of the parameters affects only x and y parameters because of $Lim_{P1}x$ and $Lim_{P1}y$, however this is not true. Experiments in chapter 7 show that limits of other parameters (e.g. z, θ) also get refined under some situations.

5.2.4 Step 3

From *step 2* of the approach we get following two important components:

1. An estimate (refined) of the limits of each parameter.
2. A set of labelled examples or instances of the action.

In *step 3* we exploit the above mentioned components to modify the planning operator such that this modification avoids the occurrence of the same *unknown external fault* again in the future. Below given description of this step first formalizes and explains different techniques and then shows the use of these techniques in achieving the goal of this step.

Assume that *step 2* generates m labelled instances of an action. Each of these instances can be written as a row vector of length n. Where, the nth component of the vector is the label of the example (i.e. 0 or 1) and the first $n-1$ components represent the parameters of `object-1` in a sequence shown in top down manner in table 5.1. We store all the labelled instances of the action in an $m \times n$ matrix, **I**. We split **I** into two matrices **C0** and **C1** based on the labels of the instances, where **C0** contains only the *negative* or '*class 0*' instances and **C1** contains only the *positive* or '*class 1*' instances. Just like **I**, each of the first $n-1$ columns of **C0** and **C1** represents values of a parameter of `object-1`. From here on, we also use the word *features* for the parameters because of the ML literature convention.

Once again, we refer to the running example used in the illustration of *step 2*. Consider, for this example we let the algorithm shown in figure 5.9 run until it generates 2500 instances for both *class 0* and *class 1*. We store the total 5000 instances in **I** and then split the matrix. Splitting **I** into **C0** and **C1**, actually divides values of each feature into two groups depending upon the label of instances. Figure 5.10 shows the distributions of the values of x, y and z features for each group for our running example. In this figure, the left column of the subplots corresponds to the distributions of the values of the features for *class 0* and the right column corresponds to the distributions of the values of the features for *class 1*. Each '$\times$' symbol in each subplot of the figure corresponds to a value of the feature plotted against the index of the instance it belongs to (after the split). It can be seen in figure 5.10 that the distributions of the values of both x and y features (i.e. first two rows of subplots) roughly resemble uniform random distributions[8] for both *class 0* and *class 1*. Whereas, the distributions of the values of z (i.e. the last row of subplots) do not resemble uniform distributions. It is clearly visible in the figure that for *class 0* the distribution of z is *skewed* towards the larger values (i.e. $z > 0.9$), whereas for *class 1* the distribution's *skewness* is towards smaller values of z. Such patterns in the distributions are also observable for other features of instances.

We exploit the observation mentioned in the previous paragraph in finding the importance of each feature (i.e. IMP_f) in the behavior of the objects. For this, we use the measures of mean ($\mu_{f_{Ci}}$), variance ($\sigma_{f_{Ci}}$), skewness ($skew_{f_{Ci}}$) and kurtosis ($kurt_{f_{Ci}}$) of the distributions of the values of each feature in both the groups (i.e. *class 0* and *class 1*). Equations 5.1 to 5.4 give the definitions of the aforementioned measures of the distributions. In these equations and the description to follow, the notations use following conventions:

- f represents a feature and f in a subscript shows that the value of the concerned variable is calculated for each feature separately.

[8]The intervals of the distributions are the limits of the features obtained from *step 2*.

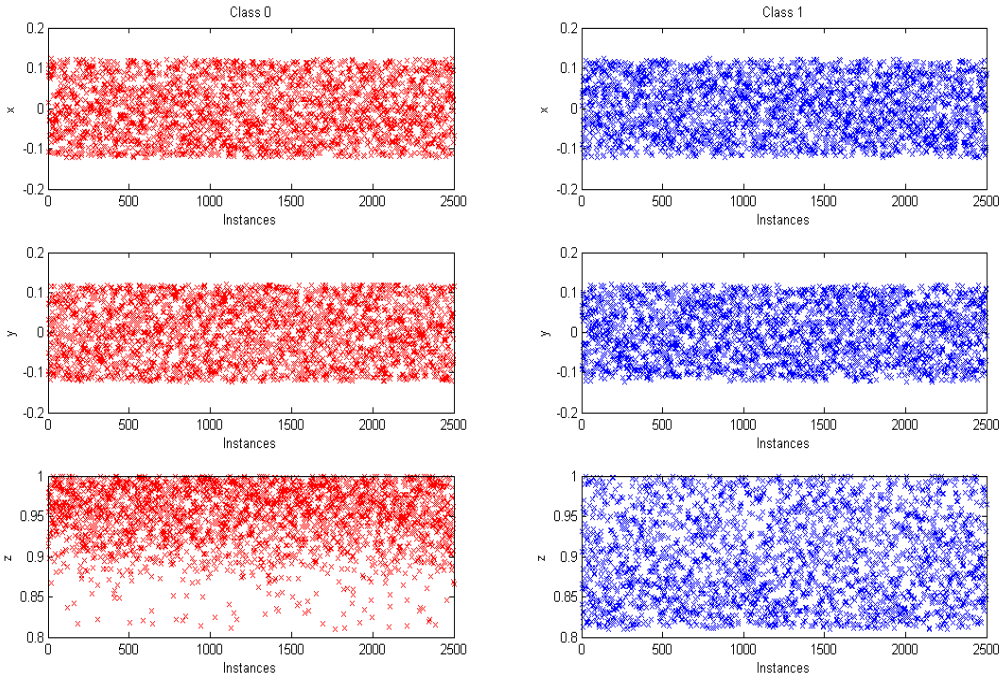


Figure 5.10: Distribution of values of x, y, z according to the labels of instances.

- A subscript i in Ci (where $i \in [0, 1]$) shows that the value is calculated separately for *class 0* and *class 1*.
- m represents the number of the rows of the matrix (i.e. **C0, C1** or **I**) concerned with the description or calculation under consideration.

$$\mu_{f_{Ci}} = \frac{\sum_{j=1}^{m} f_{Cij}}{m} \tag{5.1}$$

$$\sigma_{f_{Ci}} = \frac{\sum_{j=1}^{m} (f_{Cij} - \mu_{f_{Ci}})^2}{m} \tag{5.2}$$

$$skew_{f_{Ci}} = \frac{\frac{1}{m}\sum_{j=1}^{m} (f_{Cij} - \mu_{f_{Ci}})^3}{(\frac{1}{m}\sum_{j=1}^{m} (f_{Cij} - \mu_{f_{Ci}})^2)^{\frac{3}{2}}} \tag{5.3}$$

$$kurt_{f_{Ci}} = \frac{\frac{1}{m}\sum_{j=1}^{m} (f_{Cij} - \mu_{f_{Ci}})^3}{(\frac{1}{m}\sum_{j=1}^{m} (f_{Cij} - \mu_{f_{Ci}})^2)^2} \tag{5.4}$$

For each feature, IMP_f is calculated as the sum of the absolute differences between corresponding measures of distributions for *class 0* and *class 1*. Equations 5.5 to 5.8 and line '4' in figure 5.11 give the equations for calculating IMP_f. Using the values of IMP_f we calculate another metric for the features, which we call *weight of the feature* (i.e. W_f). W_f is simply the normalized value of IMP_f (see line '3' in figure 5.11) which represents the importance of a feature relative to other features in the instances. We consider IMP_f and W_f to be the metrics of importance of features in the *behavior* of objects because

these metrics are based on the labels of the instances. And, the labels in turn are based on the desirability of the *behavior* shown by the objects when the initial states of the objects are represented by the values of the features in the instances.

$$\Delta\mu_f = \mu_{f_{C1}} - \mu_{f_{C0}} \tag{5.5}$$

$$\Delta\sigma_f = \sigma_{f_{C1}} - \sigma_{f_{C0}} \tag{5.6}$$

$$\Delta skew_f = skew_{f_{C1}} - skew_{f_{C0}} \tag{5.7}$$

$$\Delta kurt_f = kurt_{f_{C1}} - kurt_{f_{C0}} \tag{5.8}$$

1. **function** CALCULATE-FEATURE-WEIGHTS (**C0, C1**) **returns W**
2. *for each feature, f*
3. $W_f = \frac{IMP_f}{\sum_{i=1}^{n-1} IMP_f}$, where
4. $IMP_f = abs(\Delta\mu_f) + abs(\Delta\sigma_f) + abs(\Delta skew_f) + abs(\Delta kurt_f)$
5. **W** = A vector composed of W_f for each feature
6. return **W**

Figure 5.11: Definition of the function CALCULATE-FEATURE-WEIGHT.

Figure 5.12 shows a bar chart[9] of the values of W_f for all the features for our example of the action of releasing the `bottle over` the `cube`. As can be seen in the chart, the most important feature in this action is the height (i.e. the value of z-coordinate) at which the `bottle` is released. It should be noticed, that here greater value of W_f for z implies that the final state of the `bottle` is more sensitive to z, *given* the values of all the features are chosen within the limits suggested by *step 2* of the approach. Within these limits values of other features hardly affect the behavior of the `bottle` in its final state. This result is also very intuitive. For instance, we know that *given* the `bottle` is kept approximately straight and static then it will stand `straight along z` axis `on` the `cube` if it is released `over` the `cube` from a lower height, no matter what x and y coordinates we choose for releasing it. The `bottle` *may* fall to its side (and perhaps end up on the floor) if we release it using the same conditions but increasing the height of the release.

In our settings, W_f always gives a good estimate of the relative importance of the features. The reason behind this fact is as following. We generate instances of an action by selecting values for each feature from uniform random distributions which are defined by the limits of the corresponding features. Within these limits, if the final state of the objects is sensitive to some particular range(s) of values of a feature then it is always reflected

[9]Notice the difference of the symbols used to represent the features on x-axis. For plots and charts we use these symbols for features for clarity in reading the smaller fonts. The right to left sequence of the symbols on x-axis is same as the top down sequence of table 5.1. We also use the same symbols in rest of the charts and plots shown in this thesis.

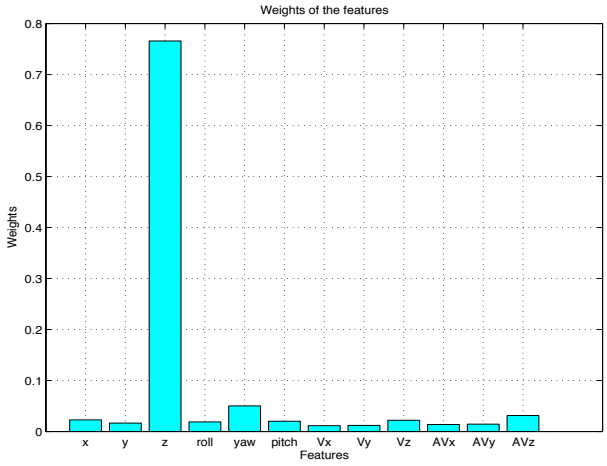


Figure 5.12: Bar chart showing the weights of the features.

in the distributions of that feature obtained by grouping its values according to class labels (e.g. figure 5.10). Otherwise, such distributions remain similar to uniform random distribution (for both *class 0* and *class 1*) because in the grouping process the values get grouped randomly. For each feature, W_f simply captures the difference in the distributions obtained after the grouping process. Hence, W_f always has a larger value for a feature that has more affect on the final state. We define the function of calculating W_f for each feature in an algorithmic form in figure 5.11.

It is possible to use the grouping of values of each feature (e.g. as in figure 5.10) in estimating a vector of features in which each feature has (approximate) best values[10]. It can be done by dividing the values of each feature into smaller *bins* and choosing a value for each feature from the bins (of the corresponding features) which strongly associate with the positive (i.e. *class 1*) instances of the action. Naturally, in selection of such values it is necessary to be more precise about the values of the important features. This precision can be obtained by increasing the number of bins (i.e. NB_f) for those features which have high W_f values. For each feature, we calculate NB_f using the formula given in equation 5.9. In this equation $Rank_f$ represents ranking of the feature, which is simply the index number of the feature when the features are arranged in the decreasing order of W_f. For instance, according to figure 5.12, z has $Rank_f = 1$; yaw (i.e. θ) has $Rank_f = 2$ and so on. Also, n represents the number of columns of $\mathbf{I}$; and $binMul$ (a positive real number) and $augBin$ (a positive integer) are constants which are selected by deciding the desired maximum and minimum numbers of bins for the features. For instance, in our current example there are 12 features (i.e. $n = 13$) and we want that the most important feature (i.e. z) has 25 bins and the least important feature (i.e. Vx or $\dot{x}$) has only 5 bins. For that we can select $binMul = 1.8$ and $augBin = 4$. This divides the limits of z into 25 bins

[10]Values of the features which are most suitable for the desired behavior of the objects.

and limits of $\dot{x}$ into 5 bins, whereas limits of all other features are divided in the number of bins between 5 and 25, depending upon their importance.

$$NB_f = floor(\frac{binMul * (n-1)}{Rank_f}) + augBin \tag{5.9}$$

In order to select the right bin of values for the (approximate) best value of a feature, we need to evaluate that how *strongly* its bins are associated with positive instances of the action. We calculate this strength of bins (i.e. $BinStrength$) using the formula shown in equation 5.10. For each feature, $BinStrength$ of each bin is the cumulative contribution of all p values of the feature in the training instances which fall in that bin.

$$BinStrength = \sum_{p} k(Ci) * W_f * S(Ci) \tag{5.10}$$

In equation 5.10, k is a function of the label of the class to which the instance (under consideration) belongs. For *class 1* (i.e. $C1$), the value of k is 1. For *class 0* (i.e. $C0$), the value is a negative ratio of the total number of $C1$ instances to the total number of $C0$ instances. Mathematically,

$$k(Ci) = \begin{cases} 1 & \text{for } C1 \\ -\frac{\#\text{of } C1 \text{ instances}}{\#\text{of } C0 \text{ instances}} & \text{for } C0 \end{cases}$$

For a single bin (of a feature), let us denote the total number of values of the feature that belongs to *class 0* instances and *class 1* instances by $pC0$ and $pC1$ respectively. The above definition of $k(Ci)$ ensures that if we measure $BinStrength$ using only $k(Ci)$ (i.e. $W_f = S(Ci) = 1$, in equation 5.10), then the $BinStrength$ will be zero for a bin that has values of $pC0$ and $pC1$ in proportion to the total number of instances of *class 0* and *class 1*, respectively. The value will be positive if $pC(i)$ favor *class 1* and negative if these numbers are in favor of *class 0*.

Assume that for our running example we have 1000 instances for $C1$ and only 200 instances for $C0$ (instead of 2500 instances for both). In other words, the division of instances is heavily *skewed*. Under these conditions, let us consider the situation of one of the five bins for $\dot{x}$ that happened to have 100 values that correspond to *class 1* instances (i.e. $pC1 = 100$) and 20 values that correspond to *class 0* instances (i.e. $pC0 = 20$). If we calculate $BinStrength$ for this bin using only $k(Ci)$, then it will be exactly zero. This, is a contrived ideal situation for a bin which should have zero $BinStrength$. Such situations can occur only rarely in the actual division of the values of features among the classes. However, it is very practical that a bin who's ideal $BinStrength$ should have been zero,

favors a particular class label by a few number of values. For instance, for the current example the division of the values in the bin can be $pC1 = 97$ and $pC0 = 23$. If we evaluate $BinStrength$ using these values, it will be -18.0. This is a very high negative strength considering the number of values for $C0$ and $C1$ deviate only a little from the ideal case. Whereas, this deviation could just be a result of noise in the data. We introduce the term $S(Ci)$ in equation 5.10 to avoid such sharp variations in the values of $BinStrength$. The formula for calculating $S(Ci)$ is given in following equation:

$$S(Ci) = 0.9 + \frac{\#\text{of } Ci \text{ instances}}{5 * \text{Total instances}} \tag{5.11}$$

If we use $S(Ci)$ along with $k(Ci)$ in equation 5.10 then for our current example the $BinStrength$ is evaluated to -4.5 instead of -18.0. $S(Ci)$ can be seen as a *scaling* factor for $BinStrength$ with its value varying within the interval $[0.9, 1.1]$[11]. $S(Ci)$ stops affecting the $BinStrength$ if the division of instances is *uniform* between *class 1* and *class 0*. That is, the total number of instances for *class 1* is same as total number of instances for *class 0*. In that case $S(C0) = S(C1) = 1$. Values of $S(Ci)$ move closer to extreme values as the division of the instances gets more skewed. This behavior of $S(Ci)$ is intentional and it reduces the sharp variations in $BinStrength$ values, which become more pronounced in the case of skewed data. The insights discussed so far about specifying the bins for the features are presented in an algorithmic form in figure 5.13.

Once the bins for the features have been specified and their strengths have been found, we can select the bins with the maximum positive strengths for each feature to choose the best values of the features. The maximum positive strength is an indicator that within the limits specified by the boundaries of the bin, majority of the values of the feature are associated with *class 1* instances. In other words, majority of these values are associated with the desired behavior of the objects. The values in the bins with the maximum negative $BinStrenght$ are strongly associated with the undesired behavior of the objects in a similar manner. Figure 5.14 shows the best bins (green boundaries) and the worst bins (black boundaries) for x, y and z features for our running example. It can be noticed in the figure that widths of the bins for x and y are quite large whereas the bin widths for z are very small. Furthermore, the bin shown for *class 1* values of z corresponds to much smaller values of z as compared to the values corresponded by *class 0* bin. This shows that by specifying the bins we can estimate the best and the worst values for the features, such that the values are more precise for the features which are important in the behavior of the objects.

For each feature we choose the best value to be the one in the middle of the bin. That is, the mean value of the limits of the bin. We compose a vector of the best values

[11] Limits of the interval are selected empirically with the help of the experiments presented in chapter 7.

1. **function** SPECIFY-BINS (**C0,C1,W**) **returns BL, BS, NB**
2. **Set** binMul, augBin
3. *for each feature, f*
4. $NB_f = floor(\frac{binMul*(n-1)}{Rank_f}) + augBin$, where
5. $Rank_f$ = Position of a feature when the features are arranged
6. in decreasing order of W_f
7. **NB** = A vector composed of NB_f for each feature
8. *for each feature, f*
9. Divide the range of the feature in equal NB_f bins
10. **BLf** = A matrix containing the limits of each bin
11. *for each bin*
12. $BinStrength = \sum_p k(Ci) * W_f * S(Ci)$, where
13. p =Total number of feature values falling in the bin
14. $S(Ci) = 0.9 + \frac{\#\text{of } Ci \text{ instances}}{5*\text{Total instances}}$
15. $k(Ci) = 1$ for $i = 1$ and $= -\frac{\#\text{of } C1 \text{ instances}}{\#\text{of } C0 \text{ instances}}$ for $i = 0$
16. **BSf** = A vector composed of strengths of each bin
17. **BL** = A matrix formed by joining **BLf** of each feature
18. **BS** = A vector formed by joining **BSf** of each feature
19. return **BL, BS, NB**

Figure 5.13: Definition of the function SPECIFY-BINS.

of each feature (i.e. **VC1**) and call it the *suggested vector*. If the suggested vector is used for the initial state of the objects, then the action that initially resulted in an *unknown external fault* can be completed successfully by the robot. This is true because, the suggested vector corresponds to the (approximate) best initial state to achieve the desired/expected behavior of the objects. Furthermore, this best state is estimated by considering as many features of the object as possible[12]. From the suggested vector, we also know the limits of each feature within which the behavior of the objects will remain the same. These limits correspond to the boundaries of the best bins for the features.

Figure 5.15 shows the overall procedure of finding the suggested vector and the corresponding limits of the bins in a matrix form (i.e. **VC1Lim**). The function that calculates **VC1** and **VC1Lim** is named N-Bins because it estimates the suggested vector by dividing the limits of each feature in n bins, where the value of n is proportional to the importance of the corresponding feature. N-Bins is a general algorithm that uses the functions presented in figure 5.11 and 5.13 for the calculations.

[12]The total number of features is only restricted by the simulator/physics engine. In our work we use all the parameters of the object that can be handled by the simulator used in the simulation process.

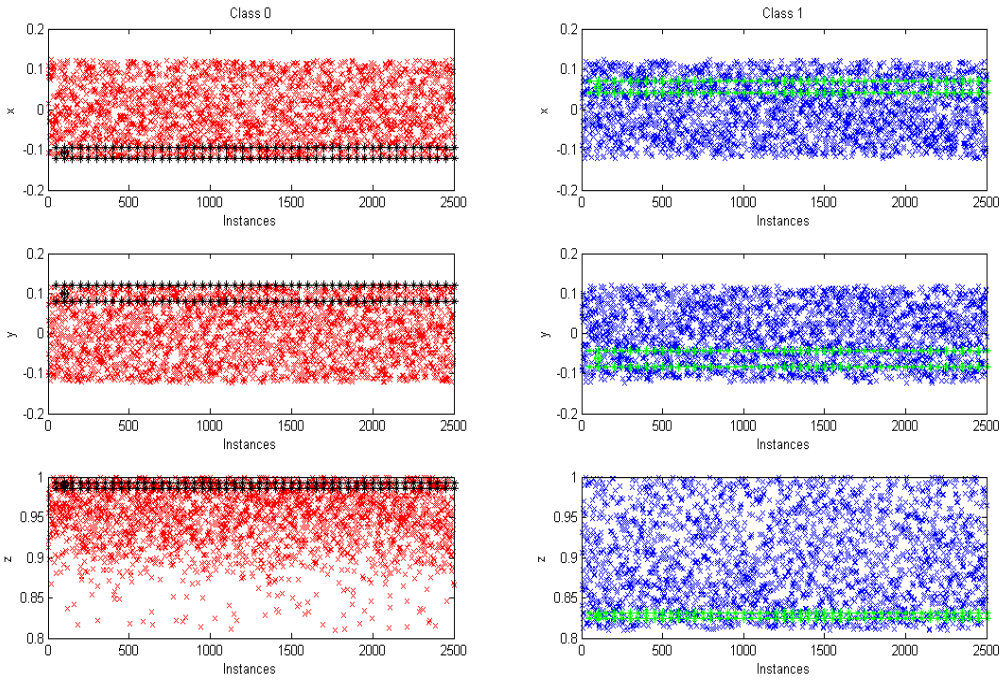


Figure 5.14: Suggested bins for x, y, z values.

1. **function** N-Bins (**I**) **returns VC1, VC1Lim**
2. **input: I**, an $m * n$ matrix composed of labelled instances
3. **C0, C1** ← Split instances according to their labels
4. **W** ← CALCULATE-FEATURE-WEIGHTS (**C0, C1**)
5. **BL, BS, NB** ← SPECIFY-BINS (**C0,C1,W**)
6. *for each feature, f*
7. Select the bin from **BL** that corresponds to maximum value in **BS**
8. $fValue$ = mean of the limits of the selected bin
9. **VC1** = Vector formed by $fValue$s of each feature
10. **VC1Lim**= Matrix formed by the corresponding limits of the selected bins
11. return **VC1**, **VC1Lim**

Figure 5.15: N-Bins as a feature vector suggester.

N-Bins as a binary classification algorithm

It is possible to convert N-Bins into a binary classification algorithm with a slight modification. This modification is shown in figure 5.16. The algorithm shown in this figure exploits the knowledge of bins and *BinStrength*s to classify a test instance as a *class 0* or *class 1* instance. Basic concept behind this classification is very simple. For any test instance, the algorithm first specifies the bins for each feature in which its value falls. Then it estimates the strength of the overall vector of the features (i.e. the test instance) by cumulative strength of the bins specified for each feature. If the value for this strength is greater than 0, then the instance is predicted to be a *class 1* instance, otherwise it is predicted to be a *class 0* instance. In finding the cumulative strength of the feature vector (i.e. line 11 in the figure), the algorithm has to take care of the fact that each feature has different number of bins (NB_f). Therefore, it scales the bin strength corresponding to each feature (i.e. $BinStrength_f$) with a factor that represents the proportion of the bins for that feature in total number of bins for an instance.

1. **function** N-Bins (**I, T**)
2. **input: I**, an $m * n$ matrix composed of labelled instances
3. **T**, an $r * n$ matrix composed of labelled test instances
4. **C0, C1** ← Split instances according to their labels
5. **W** ← CALCULATE-FEATURE-WEIGHTS (**C0, C1**)
6. **BL, BS** ← SPECIFY-BINS (**C0,C1,W**)
7. *for each test instance* **t**, *in* **T**
8. *for each feature, f of* **t**
9. Select the bin from **BL** in which the value of the feature falls
10. Select the corresponding *BinStrength* from **BS**
11. *if* $\sum_f BinStrength_f * \frac{NB_f}{\sum_f NB_f} > 0$
12. *then* $predictedLabel = 1$
13. *else* $predictedLabel = 0$

Figure 5.16: N-Bins as a binary classifier.

Modifying the planning operator

The N-Bins algorithm shown in figure 5.15 provides a robot with the ingredients of performing an action reliably[13] against *unknown external faults*. That is, it provides values for the parameters of objects (i.e. vector **VC1**) which result in successful completion of action and it also provides the information (i.e. matrix **VC1Lim**) that within what ranges of these values the outcome of the action is going to remain the same. A simple way to increase the reliability of an action using these ingredients is to let the predicate

[13]Under the assumption that the models of the objects used in the simulation process of generating the instances are accurate enough. And, the used simulator can replicate the natural behavior under natural physics laws.

`Allowed/3` (see figure 5.1), in the *preconditions* of the planning operator, be *true* only under one of the following two conditions:

- *Condition-1*: The robot releases `object-1` according to the exact values of **VC1**.
- *Condition-2*: The robot releases `object-1` with the values of each of its parameters within the ranges defined by **VC1Lim**.

Although theoretically it is possible for a robot to meet any of the conditions stated above, but practically it can be problematic because of physical constraints of the environment or the robot itself. For example, it can happen that x or y value suggested by **VC1** is beyond maximum possible extension of the robot's manipulator. Therefore, we introduce another condition which if satisfied also allows the robot to perform the action reliably. This condition is stated below.

- *Condition-3*: The robot releases `object-1` with the values of each parameter selected such that a classification algorithm classifies the test instance composed of those values as *class 1* instance.

This condition is much weaker than the first two and should only be used if the first two conditions are physically impossible to satisfy. As mentioned in section 2.1, there are many ML algorithms which can be used for classification and hence, for evaluation of *Condition-3*. Since N-Bins can also be used as a binary classification algorithm, therefore it can also be used to evaluate *Condition-3*. When `Allowed/3` is evaluated *true* based on *Condition-3* then it is important that the algorithm chosen for classification has high accuracy.

It is also possible to define a condition for `Allowed/3` that is more strict than *Condition-3* but less strict than *Condition-2*. That is:

- *Condition-2a*: The robot releases `object-1` with the values for first n (where n is an integer who's value is subjective to circumstances) important parameters within the ranges defined by **VC1Lim** and the instance composed of the values of all the parameters is classified as *class 1* instance by the classification algorithm.

It is possible to create more of such conditions with the help of the information provided by (all of the) bins and their strengths. This opportunity arrises primarily from the representation of the data in terms of bins in the N-Bins algorithm. As will be seen in chapter 7, this representation also provides robustness against noise in the data.

Based on the discussion above, we can define `Allowed/3` as following:

`Allowed(action, object-1, object-2)` $\Longleftrightarrow$
`Condition-1` $\vee$ `Condition-2` $\vee$ ..... $\vee$ `Condition-n`.

This definition of `Allowed/3` is generic and only theoretical. Furthermore, it assumes that the conditions in the body of the above FOL sentence are evaluated from left to right because it is assumed that the conditions are arranged from strict to relaxed in this direction. In this work we do not commit to a definite definition of `Allowed/3`, because it possible to create many conditions for the body of the above sentence. In our experiments with the proposed approach (presented in chapter 7) we concentrate only on the aspects related to *Condition-1, 2* and *3*.

The approach presented in this chapter is formulated as a three-step scheme. Above explanation of this scheme concentrates on all the major aspects of the approach. There are still some minor issues regarding the approach which are not discussed in this chapter to keep the explanation simple. We highlight these issues in an illustrative manner in chapter 7. The above explanation of the scheme also ignores one major component of our work which comprises the definitions of the predicates used in finding the simulation description. Collectively, we refer to these predicates as *description vocabulary*. The *description vocabulary* and the *initial limits* of the parameters are the topic of discussion of the next chapter. In this work we do not explain the approach as a single algorithm or a detailed schematics that covers all the aspects of the approach. This is because the basic scheme of the approach is straightforward (as shown in figure 5.2). However, it becomes to *appear* unnecessarily complex if we try to collect the information in a single schematics or algorithm. Therefore, in this thesis we focus on the descriptive explanation of the work and organize this explanation (in this chapter and chapter 7) according to the basic schematics given in figure 5.2. This also allows us to concentrate on the major aspects of our work while treating them independently from each other.

6 Description vocabulary

In *step 1* of the approach discussed in chapter 5, we find description of the behavior of the objects that are simulated. This description is found as FOL sentences that are conjunctions of predicates. Each of these predicates signifies a particular aspect of the objects under consideration. In this chapter of the thesis we formalize the definitions of these predicates. The formal definitions of the predicates state the conditions for each predicate to be true. Since each of the predicates defined in this chapter is used in describing the behavior of the objects, therefore we refer to the predicates collectively as *description vocabulary*. In the approach presented in chapter 5 the *description vocabulary* is also utilized in labeling the training instances generated in the example generation process.

The *description vocabulary* consists of 67 predicates in total. In this chapter these predicates are defined in different sections. The division of sections is according to the aspects of the objects covered by the predicates. The only exception is section 6.1, which formalizes the RCC-8 relations (discussed in chapter 2) in a manenr that is suitable for our approach. These RCC-8 relations are later used in section 6.2 to cover the aspect of *connectedness* of the objects.

Settings

The definitions of the predicates in the *description vocabulary* assume the underlying topological space of the world to be in $\mathbf{R}^3$. Objects in the world are considered to occupy a subset of the space. Any object is composed of its interior and its boundary surface. Shapes of the objects are defined by the closure of their interiors. This implies that it is possible for two objects to share points that are common on their boundaries. Whenever the definitions of the predicates use the settings other than above, we explicitly mention these settings in the related section of this chapter.

Conventions

The *description vocabulary* consists of definitions of predicates only. However, in order to formally define the predicates we also make use of logical functions. These functions are used in the logical definitions of the predicates and we also give formal definitions for each of these functions. The definitions of the predicates and the functions observe following conventions.

1. Definition of each function and predicate is enumerated. The number assigned to the definition of each predicate that is a part of the *description vocabulary* is given as '*6.definition-number*'. Whereas, the number assigned to the definitions of

the predicates and functions which are not (primarily) a part of the *description vocabulary* is given as '*6.section-number.defintion-number*'.

2. Definitions of functions are given in simple english only, whereas the definitions of predicates are also stated in logical expressions. These expressions make use of other functions and predicates defined in this chapter.

3. Existence of universal quantifiers in the logical expression is understood but not explicitly mentioned.

4. Logical terms follow the conventions of FOL in representing variables, constants, predicates and functions. That is, the first letter of variables is kept smaller whereas, the first letter of a constant is kept capital. Similarly, the first letter in the name of a function is smaller, whereas the first letter in the name of a predicate is capital.

5. A double letter variable corresponds to a set of the entities represented by a single letter variable. For instance, the definitions use variable `p` to represent a point in the space and `pp` to represent a *set* of points in the same space. We use numeric values in the names of the variables only to distinguish the variables of the same type. For example, `pp1` represents one set of points, whereas `pp2` represents another.

6. In the definitions, any deviation from the conventions given above is explicitly mentioned.

In this chapter we also give details about the limits of the parameter. In our approach, these limits are found directly from the predicates that describe the initial states of the objects. The limits of the parameters given in this chapter are termed as the *initial limits* in other chapters of the thesis. This chapter does not follow any particular conventions while giving details about the limits. We use different techniques to clarify the values of the limits for different parameters. We avoid unnecessary explanations of the concepts whenever the concepts are clear from the context. In this chapter we also give the details about the implementation of the predicates. We do this by discussing only the important issues that require explanation.

6.1 RCC-8 relations

In this section we redefine the RCC-8 relations mentioned in section 2.4.1 of chapter 2. Although, the concepts represented by the new definitions are exactly the same as those in chapter 2, but the definitions themselves differ alot. The differences in the definitions is because of following two reasons:

1. For practical purposes we consider the regions in $\mathbf{R}^2$ to be consisting of limited number of points which are sparsely distributed over the referred region in a random fashion.

2. The definitions use functions and other predicates instead of directly using sets of points (as in section 2.4.1). This representation is based on the implementation of the RCC-8 relations in this work.

We consider regions to be represented by limited number of points in order to make evaluation of the relations computationally effective. However, this effectiveness comes at the cost of complexity in the definitions of the relations. In order to keep the definitions simple we make use of definitions of some auxiliary predicates and functions. The knowledge represented by each of such definition can directly be implemented as a computer program. This allows us to implement the definitions of RCC-8 relations in a systematic and computationally effective manner. Below we give the definitions of the functions and predicates used by RCC-8 relations.

Definition 6.1.1: Function `count-Points(pp)` refers to the number of points contained in a set of points `pp`.

Definition 6.1.2: Function `boundary-Points(pp1, pp2)` refers to a subset of `pp1` that consists of all the points which exist on the boundary of the closed surface represented by the set of points `pp2`.

Definition 6.1.3: Function `internal-Points(pp1, pp2)` refers to a subset of `pp1` that consists of all the points which exist in the interior of the closed surface represented by the set of points `pp2`.

Definition 6.1.4: Function `external-Points(pp1, pp2)` refers to a subset of `pp1` that consists of all the points which exist in the exterior of the closed surface represented by the set of points `pp2`.

Implementation explanation (6.1.2 - 4)

In the functions 6.1.2 - 4, the sets of points `pp1` and `pp2` correspond to `object-1` and `object-2` respectively, in our settings. In our implementation, each point `p1` in `pp1` is represented by x and y coordinates measured in the frame of reference attached to `object-2`. Each `p1` itself corresponds to a vector $\mathbf{v}_{obj1}$ of a marker on `object-1` (i.e. $\mathcal{M}_{obj1}$, see section 5.2.2). This implies that the set `pp1` can be seen as a set of xy-plan projections (where $z = constant$) of the markers of `object-1`. Same is true for each point `p2` in `pp2` for `object-2`. In our implementation we use 50 markers per $100cm^2$ surface area of `object-1`. With such a number, the convex hull of `pp1` gives a good approximation of xy-plane projection of the shape of `object-1`. In case of `object-2`, we use `pp2` to calculate the exact projection of the shape of the concerned surface of `object-2` with the

help of g_{obj2} of each marker. Thus, we can say that in our settings the domain of the variables of the functions are closed planner surfaces formed by xy-plan projections of the objects.

In our implementation we define the boundary of the surface of `object-2` to be $2mm$ wide. We find the sets of points referred by each function with the help of the markers of objects. We simplify the geometric calculations with the help of the information available in g_{obj2}. These calculations are kept precise down to $1mm$.

Definition 6.1.5[1] : Predicate `EvenBoundaryDist(pp1, pp2)` is true if those points in `pp1` which lay on the boundary of the closed surface represented by `pp2`, are approximately evenly distributed over the complete boundary of the said surface.

Implementation explanation (6.1.5)

Implementation of the predicate `EvenBoundaryDist/2` uses the set of points referred by `boundary-Points/2`. It also uses the information about the shape of the closed surface represented by `pp2`. This information is available in g_{obj2} of each $\mathcal{M}_{obj2}$. In order to evaluate the truth value of `EvenBoundaryDist/2` we adopt a simple strategy. That is, we divide the *boundary* of the surface represented by `pp2` into different segments and then check that whether or not the numbers of points of `pp1` laying on each segment of the boundary are in proportion to the length of the corresponding segments. If these numbers are in proportion to the lengths of the segments then the predicate `EvenBoundaryDist/2` is evaluated true, otherwise it is evaluated false.

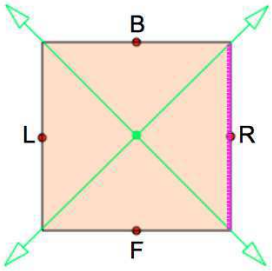


Figure 6.1: Segmentation of the boundary of a surface represented by `pp2`.

For the segmentation of the boundary, we make use of vonoroi decomposition of the closed surface represented by `pp2`. For example, figure 6.1 shows a surface (i.e. a square) that can be represented by the points of `pp2`. We decompose this surface with the help of four vonoroi regions corresponding to the four markers (shown as R, B, L and L) on the boundaries of the surface. The rays in green color show the boundaries of the regions after the decomposition. Each segment of the outer boundary of the surface, that is

[1] We do not give logical expression for this definition of a *predicate* because of the detailed implementation explanation.

in between two rays, belongs to one marker. For instance, the segment shown in magenta color belongs to marker R. Let us denote the length of any of such segment by $l_{\mathcal{M}}$ and the fraction of the boundary of the surface that is covered by $l_{\mathcal{M}}$, by $f_{\mathcal{M}}$. If for any $l_{\mathcal{M}}$ there are $n_{\mathcal{M}}$ points of `pp1` that lay on that segment, then in our implementation the predicate `EvenBoundaryDist/2` evaluates to true only if, for each boundary segment:

$$n_{\mathcal{M}} \geq 0.7 \times f_{\mathcal{M}} \times \text{total points referred by } \texttt{boundary-Points/2} \tag{6.1}$$

6.1 is an inequality instead of being an equation because of the fact that the points of `pp1` are projections of randomly distributed markers that are also sparse. It is because of the same reason that the right hand side of 6.1 includes a factor 0.7 in it. This means that we allow the predicate `EvenBoundaryDist/2` to be true even if $n_{\mathcal{M}}$ is as low as 70% of the points that should ideally be laying on the segment. In order to check the inequality 6.1, we need to calculate $f_{\mathcal{M}}$ for all the segments that can be formed for the geometries we consider in this work. These calculations are given in *appendix (B)*. In these calculation we systematically find $f_{\mathcal{M}}$ with the help of $l_{\mathcal{M}}$ for each possible boundary segment of the geometries.

Below are the definitions of RCC-8 relations that we use in our implementation. These definitions make use of the functions and predicates discussed above.

Definition 6.1.6: Predicate `DC(pp1, pp2)` is true when the convex hull of all the points in `pp1` is disconnected from the closed planner surface represented by `pp2`.
`DC(pp1, pp2)` $\Leftrightarrow$
`(count-Points(interior-Points(pp1, pp2))` $= 0) \wedge$
`(count-Points(boundary-Points(pp1, pp2))` $= 0) \wedge$
`(count-Points(pp1)` $=$ `count-Points(exterior-Points(pp1, pp2))).`

Definition 6.1.7: Predicate `EC(pp1, pp2)` is true when the convex hull of all the points in `pp1` is externally connected with the closed planner surface represented by `pp2`.
`EC(pp1, pp2)` $\Leftrightarrow$
`(count-Points(interior-Points(pp1, pp2))` $= 0) \wedge$
`(count-Points(exterior-Points(pp1, pp2))` $> 0) \wedge$
`(count-Points(boundary-Points(pp1, pp2))` $> 0)$.

Definition 6.1.8: Predicate `TPP(pp1, pp2)` is true when the convex hull of all the points in `pp1` is a tangential proper part of the closed planner surface represented by `pp2`.
`TPP(pp1, pp2)` $\Leftrightarrow$
`(count-Points(interior-Points(pp1, pp2))` $> 0) \wedge$
`(count-Points(exterior-Points(pp1, pp2))` $= 0) \wedge$

(count-Points(boundary-Points(pp1, pp2)) $> 0) \wedge$
$\neg$EvenBoundaryDist(pp1, pp2).

Definition 6.1.9: Predicate NTPP(pp1, pp2) is true when the convex hull of all the points in pp1 is a non tangential proper part of the closed planner surface represented by pp2.
NTPP(pp1, pp2) $\Leftrightarrow$
(count-Points(exterior-Points(pp1, pp2)) $= 0) \wedge$
(count-Points(boundary-Points(pp1, pp2)) $= 0) \wedge$
(count-Points(pp1) = count-Points(interior-Points(pp1, pp2))).

Definition 6.1.10: Predicate PO(pp1, pp2) is true when the convex hull of all the points in pp1 is partially overlapping the closed planner surface represented by pp2.
PO(pp1, pp2) $\Leftrightarrow$
(count-Points(interior-Points(pp1, pp2)) $> 0) \wedge$
(count-Points(exterior-Points(pp1, pp2)) $> 0) \wedge$
((count-Points(boundary-Points(pp1, pp2))$= 0) \wedge$ EvenBoundaryDist(pp1,pp2))$\vee$
(count-Points(boundary-Points(pp1, pp2)) $> 0) \wedge \neg$EvenBoundaryDist(pp1,pp2))).

Definition 6.1.11: Predicate EQ(pp1, pp2) is true when the convex hull of all the points in pp1 is approximately equal to the closed planner surface represented by pp2.
EQ(pp1, pp2) $\Leftrightarrow$
(count-Points(interior-Points(pp1, pp2)) $> 0) \wedge$
(count-Points(exterior-Points(pp1, pp2)) $= 0) \wedge$
(count-Points(boundary-Points(pp1, pp2)) $> 0) \wedge$ EvenBoundaryDist(pp1, pp2).

Definition 6.1.12: Predicate NTPPi(pp1, pp2) is true when the convex hull of all the points in pp1 is a non tangential proper part of the closed planner surface represented by pp2.
NTTPi(pp1, pp2) $\Leftrightarrow$
(count-Points(interior-Points(pp1, pp2)) $> 0) \wedge$
(count-Points(exterior-Points(pp1, pp2)) $> 0) \wedge$
(count-Points(boundary-Points(pp1, pp2)) $> 0) \wedge$ EvenBoundaryDist(pp1, pp2).

Definition 6.1.13: Predicate TPPi(pp1, pp2) is true when the convex hull of all the points in pp1 is a tangential proper part of the closed planner surface represented by pp2.
[2]TTPi(pp1, pp2) $\Leftrightarrow$
$\neg$DC(pp1, pp2)$\wedge$ $\neg$EC(pp1, pp2)$\wedge$ $\neg$TPP(pp1, pp2)$\wedge$ $\neg$NTPP(pp1, pp2)$\wedge$
$\neg$PO(pp1, pp2)$\wedge$ $\neg$EQ(pp1, pp2)$\wedge$ $\neg$NTPPi(pp1, pp2).

[2]This defintions exploits the fact that RCC-8 relations are JEPD.

6.2 Connectedness

The RCC-8 relations are primarily the relations which cover the aspect of *connectedness* of two regions. In context of QSR, the domain of the variables of these binary relations are continuous regions (usually in $\mathbf{R}^3$). However, in this work we use RCC-8 relations in a different manner. In our work, we derive 24 different relations with the help of RCC-8 relations and use them to cover the aspect of *connectedness* of the regions (i.e. the objects). There are two major reasons for us to deviate from the normal practice:

1. In our work we do not use the relations for reasoning purpose as in QSR. We use the relations (i.e. the predicates) only to capture the state of the objects. Therefore, in order to capture the state of the objects at a better granularity level we derive further relations from RCC-8 relations.

2. In QSR, RCC-8 relations do not assume existence of reference frame(s). However, this is not the case in our work. Since reference frames can (always) be assumed in robotic manipulation tasks, we take advantage of their presence in defining the predicates in this work. Availability of reference frames (and hence numeric values of the coordinates of the objects) allows us to find computationally effective definitions of the predicates which cover the aspect of *connectedness* of the objects.

The domain of the variables in RCC-8 relations defined in section 6.1 are the regions in $\mathbf{R}^2$. These regions or surfaces are given as sets of points on xy-plane. The relations defined in this section are derived by adding the third dimension to the regions (i.e. z-coordinates of the points on objects). Here, we attach a reference frame to the surface defined by `pp2` such that $z = 0$ for this surface. That means, any point with $z > 0$ is higher than this surface and a point with $z < 0$ is lower than it. The 24 relations defined in this section evaluate the truth values of RCC-8 relations between the xy-plane projections of the objects while taking care of the relative location of the objects along the z-axis of the reference frame discussed above. These relation also make use of auxiliary functions which are defined below in definitions $6.2.1 - 4$.

Definition 6.2.1: Function `xy-projection(pp)` refers to the set of points in $\mathbf{R}^2$ obtained by considering only the x and y coordinates of each point in $\mathbf{R}^3$ in the set of points `pp`.

Definition 6.2.2: Function `lowest(pp)` refers to the lowest point in the set of points `pp`. The lowest point is the one with the smallest z-coordinate value. Here, the points in `pp` are in $\mathbf{R}^3$.

Definition 6.2.3: Function `height(pp)` refers to the height of the highest point in the set of points `pp`. Here, again each point in `pp` is in $\mathbf{R}^3$. In this definition, the height of a set of points is denoted by the z-coordinate value of the highest point, which can also be a negative real number.

Definition 6.2.4: Function `z-coordinate(p)` refers to the value of the z-coordinate of a point `p`.

Below are the definitions and explanation of the 24 predicates that cover the aspect of connectedness of the objects. In these definitions the objects are represented as sets of points in $\mathbf{R}^3$.

Description vocabulary for connectedness of objects

Definition 6.1: Predicate `Above(object-1,object-2)` is true when all the points of `object-1` are higher than all the points on `object-2` and the convex hull of the xy-plane projections of each marker of `object-1` and xy-plane projection of the surface of `object-2` are disconnected.

`Above(object-1, object-2)` $\Longleftrightarrow$
`(z-coordinate(lowest(object-1))` $>$ `height(object-2))` $\wedge$
`DC (xy-projection(object-1), xy-projection(object-2)).`

Definition 6.2: Predicate `Along(object-1,object-2)` is true when the height of the lowest point of `object-1` is same as the height of the points on `object-2` and the convex hull of the xy-plane projections of each marker of `object-1` and xy-plane projection of the surface of `object-2` are disconnected.

`Along(object-1, object-2)` $\Longleftrightarrow$
`(z-coordinate(lowest(object-1))` $=$ `height(object-2))` $\wedge$
`DC (xy-projection(object-1), xy-projection(object-2)).`

Definition 6.3: Predicate `Below(object-1,object-2)` is true when the height of the lowest point of `object-1` is smaller than the height of the points on `object-2` and the convex hull of the xy-plane projections of each marker of `object-1` and xy-plane projection of the surface of `object-2` are disconnected.

`Below(object-1, object-2)` $\Longleftrightarrow$
`(z-coordinate(lowest(object-1))` $<$ `height(object-2))` $\wedge$
`DC (xy-projection(object-1), xy-projection(object-2)).`

The definitions $6.1 - 3$ capture the states of the objects when the xy-plane projections of the objects are *disconnected* from each other. For other seven relations of RCC-8, we can define 21 more predicates in a manner similar to the definitions $6.1 - 3$. In the definitions of such predicates we only need to replace `DC/2` with other RCC-8 relations. Therefore,

instead of explicitly stating the definitions of all the 24 predicates, below we give one general logical expression for all the 24 predicates.

Predicate(object-1, object-2) $\Longleftrightarrow$
(z-coordinate(lowest(object-1)) *comparison-symbol* height(object-2)) $\wedge$
RCC-8 (xy-projection(object-1), xy-projection(object-2)).

This expression gives the definitions of all the 24 *Predicate*s who's names are mentioned in table 6.1 (in addition to RCC-8 predicates). Definition of any of these predicates can be created by placing the corresponding *comparison-symbol* and the name of *RCC*-8 relation in the expression above.

RCC-8	*comparison-symbol* '>'	*comparison-symbol* '='	*comparison-symbol* '<'
DC	Above	Along	Below
EC	Above-EC	Along-EC	Below-EC
TPP	Tan-over	Tan-on	Tan-under
NTPP	Over	On	Under
PO	Partially-over	Partially-on	Partially-below
EQ	Over-covered	On-covered	Under-covered
TPPi	Tan-circum-over	Tan-circum-on	Tan-circum-under
NTPPi	Circum-over	Circum-on	Circum-under

Table 6.1: Definitions of predicates for the connectedness of the objects.

The 24 predicates mentioned in table 6.1 show few interesting properties. Firstly, in our settings these predicates are JEPD for any two objects in a three dimensional space. This is because we derive these predicates from RCC-8 relations for planner regions and consider all three possibilities of the relative heights of the objects. Secondly, in our settings these predicates cover the aspect of *connectedness* between two objects in a three dimensional space using only the randomly distributed markers over the surface of the objects. In comparison to considering the complete geometry of the objects to find out about their *connectedness*, implementing these predicates is much simpler. Furthermore, as will be seen in section 6.6, these predicates can also be used in defining further predicates.

In the experiments presented in chapter 7, not all of the 24 predicates discussed in this section are used. However, definition of each of these predicates has been implemented and tested. In testing the correctness of these definitions we use different models of object-1[3]. These models are also created by different combinations of cube(s), cylinder(s) and sphere(s). Top surfaces of the models for object-2 are the same as shown in figure 5.4. The empirical value of the density of the markers for object-1 (i.e. 50 markers per $100cm^2$) and the fraction 0.7 in equation 6.1, are selected as a result of the

[3]Examples of some of these models can be seen in *appendix (C)*.

testing carried out for these relations. We do not give any detailed explicit discussion on testing of all of the 24 predicates shown in table 6.1 because it is not the main issue of discussion in the thesis. It is expected that correctness and completeness of the definitions in capturing the abstract concepts behind them is already visible to the reader through the systematic explanation in the current and the previous section of this chapter.

Limits on parameters in Δ

As mentioned in chapter 5, we do not place any limits on the values of parameters x and y because of the `P1` predicate that represents the aspect of *connectedness* of objects in the simulation description. We only place the limits on z parameter by measuring the relative height of the objects. We use following rules to find these limits:

- If `z-coordinate(lowest(object-1))` $>$ `height(object-2)` then the interval of the limits of z is (h_{obj2} + h_{obj1}, m).
- If `z-coordinate(lowest(object-1))` $<$ `height(object-2)` then the interval of the limits of z is (h_{obj1}, h_{obj2}).
- If `z-coordinate(lowest(object-1))` $=$ `height(object-2)` then the interval of the limits of z is [-s, s].

In above given intervals, h_{obj2} is the height of the top surface of `object-2`; h_{obj1} is the height of the geometric center of `object-1` measured from its bottom; 'm' is the value of z in $\boldsymbol{\mu}$ in the sample behavior and s = 0.0001.

6.3 Relative direction

In this section we define the predicates that cover the aspect of *relative direction* of the objects involved in an action of a robot. Just as in section 6.2, we attach a frame of reference to the top surface of `object-2`. The origin of this frame coincides with the geometric center of the surface. In the definitions of the predicates formalized here, the objects are considered to be continuous regions in $\mathbf{R}^3$. This difference of treatment of objects (as compared to section 6.2) is due to the fact that here the concepts represented by the predicates can be defined by treating the objects as single points in the topological space. Whereas, in section 6.2 the concepts require that the objects are treated as sets of points. Below we give the definitions of the auxiliary functions for the aspect of *relative direction* of the objects. In each of these definitions the coordinates of the the points are assumed to be measured in the aforementioned frame of reference.

Definition 6.3.1: Function `center-of-Mass(object)` refers to a point $\mathtt{c}_m$ in $\mathbf{R}^3$ that represents the center of mass of the `object`.

Definition 6.3.2: Function `x-coordinate(p)` refers to the x-coordinate of point `p`.

Definition 6.3.3: Function `y-coordinate(p)` refers to the y-coordinate of point `p`.

Description vocabulary for relative direction of objects

Definition 6.25[4]: Predicate `Nn(object-1, object-2)` is true when both the x and y coordinates of the center of mass of `object-1` are negative, when measured in the frame of reference attached to `object-2`.

$$\texttt{Nn(object-1, object-2)} \Longleftrightarrow$$
$$\texttt{(x-coordinate(center-of-Mass(object-1))} < 0 \wedge$$
$$\texttt{(y-coordinate(center-of-Mass(object-1))} < 0.$$

Definition 6.26: Predicate `Nz(object-1, object-2)` is true when the x coordinate of the center of mass of `object-1` is negative and the y coordinate is equal to 0, as measured in the frame of reference attached to `object-2`.

$$\texttt{Nz(object-1, object-2)} \Longleftrightarrow$$
$$\texttt{(x-coordinate(center-of-Mass(object-1))} < 0 \wedge$$
$$\texttt{(y-coordinate(center-of-Mass(object-1))} = 0.$$

Definition 6.27: Predicate `Np(object-1, object-2)` is true when the x coordinate of the center of mass of `object-1` is negative and the y coordinate is positive, as measured in the frame of reference attached to `object-2`.

$$\texttt{Np(object-1, object-2)} \Longleftrightarrow$$
$$\texttt{(x-coordinate(center-of-Mass(object-1))} < 0 \wedge$$
$$\texttt{(y-coordinate(center-of-Mass(object-1))} > 0.$$

From the definitions 6.25 - 27, it is clear that the predicates are simply stating the three possibilities of the values of y-coordinate of the center of mass of `object-1`, while keeping the values of the x-coordinate to be negative. This fact is also depicted in the names of the predicates, where the first letter corresponds to the value of x-coordinate of the point and the second one corresponds to the y-coordinate. If we allow the values of x-coordinate to take on all three possible values (i.e. -ve, 0.0 and +ve) then there are nine possible combinations of the values of the coordinates of $\mathtt{c}_m$ of `object-1`. In this work, these combinations form the description vocabulary for the relative direction of the objects. Instead of stating all the nine combinations in logical expressions, we give these combinations in table 6.2. In this table, labels of the columns give the names of the predicates, whereas the values of x and y in each column correspond to the values of the

[4]Index number 6.1 - 6.24 are reserved for predicates shown in table 6.1.

c_m *coordinates*	Nn	Nz	Np	Zn	Zz	Zp	Pn	Pz	Pp
x	-ve	-ve	-ve	0.0	0.0	0.0	+ve	+ve	+ve
y	-ve	0.0	+ve	-ve	0.0	+ve	-ve	0.0	+ve

Table 6.2: Definitions for the predicates of relative directions of objects.

coordinates of the center of mass of `object-1` measured in the frame of reference attached to `object-2`. From left to right (i.e. from `Nn` to `Pp`), the labels of the columns correspond to indices 6.25 to 6.33 of definitions. It should be noticed that the aspect covered by these definitions is termed *relative direction* instead of *relative location* because instead of utilizing the exact locations of centers of masses of objects we only use the knowledge of quadrants and axes of the frame of reference attached to `object-2`. The aspect of relative location is implicitly covered by these definitions and the definitions for *connectedness* of the objects.

Implementation explanation (6.25 - 6.33)

Implementation of the definitions of the predicates for relative directions of the objects is straight forward. However, it is worth mentioning that we make use of approximate values in evaluating the truth values of these predicates. That means, in our evaluation we make following assumptions:

- 0.0 corresponds to an interval $[-s, s]$.
- $-$ve corresponds to an interval $(-\infty, -s]$.
- $+$ve corresponds to an interval $[s, \infty)$.

We keep $s = 0.0001$ in our experiments.

Limits on parameters in Δ

When any of the predicates from definitions 6.25 - 33 is true, it can be used to estimate the limits of x and y parameters in Δ. This is because each of the said predicate can only be true within certain limits of x and y-coordinates of the center of mass of `object-1`, as measured in the frame of reference attached to `object-2`. Furthermore, these coordinates are the only parameters on which the truth values of the predicates 6.25 - 33 depend. Limits on x and y parameters because of the predicates related to relative direction are given in table 6.3. In this table s is same as above (i.e. $s = 0.0001$).

In table 6.3 an '∞' sign corresponds to the maximum values of x or y that can be considered by the robot in performing a task. We find these values from the corresponding dimension of the concerned surface of `object-2`. For example, if a robot wants to place a bottle on a cube who's top surface has a dimension $20(cm) \times 30(cm)$; then for $Lim_{P2}x$ we replace '∞' by 12.5 and for $Lim_{P2}y$ we replace '∞' by 18.75. In these values 12.5 is calculated by finding a marker $\mathcal{M}_{obj2}$ farthest along x-axis from the origin

of the reference frame of `object-2` and taking 125% of the value of the x-coordinate of that marker. That is, $125\% \times 10 = 12.5$. Similarly, we have $125\% \times 15 = 18.75$ for $Lim_{P2}y$. For any `object-1`, we choose maximum values of x and y to be 125% of the corresponding dimension of `object-2`'s surface because we assume that the robot posses the understanding of attaining proximity to `object-2` before releasing `object-1` over it.

Predicates	$Lim_{P2}x$	$Lim_{P2}y$
`Nn`	$(-\infty, s)$	$(-\infty, s)$
`Nz`	$(-\infty, s)$	$(-\infty, \infty)$
`Np`	$(-\infty, s)$	(s, ∞)
`Zn`	$(-\infty, \infty)$	$(-\infty, s)$
`Zz`	$(-\infty, \infty)$	$(-\infty, \infty)$
`Zp`	$(-\infty, \infty)$	(s, ∞)
`Pn`	(s, ∞)	$(-\infty, s)$
`Pz`	(s, ∞)	$(-\infty, \infty)$
`Pp`	(s, ∞)	(s, ∞)

Table 6.3: Limits imposed on x and y by `object-1`'s relative direction to `object-2`.

6.4 Orientation

Definitions 6.34 - 44 formalize the concepts related to the orientation of the objects. The predicates denoted by these definition are unary predicates and the domain of the variables of these predicates is `object-1`. The definitions make use of auxiliary functions (i.e. definitions 6.4.1 - 6.4.3) which treat the objects as sets of vectors in $\mathbf{R}^3$.

Definition 6.4.1: Function `dot-product(v1, v2)` refers to the result of vector dot product of the vectors `v1` and `v2`.

Definition 6.4.2: Function `unit-vector(object, axis)` refers to a unit vector along a particular `axis`. If the `object` represents `object-2` in this function then the vector is simply a unit vector along the `axis` of the reference frame attached to `object-2`. In case the `object` represents `object-1`, then this function refers to a unit vector that is a transformation of the unit vector along an `axis` in `object-1`'s reference frame into `object-2`'s reference frame.

Definition 6.4.3: Function `object-type(object)` refers to a constant symbol `NonDirectional`, if the maximum length; maximum height and maximum width of an object are equal. The function refers to a constant symbol `Directional` if the aforementioned condition is not true.

Description vocabulary for orientation of `object-1`

Definition 6.34: Predicate `StraightAlong-x(object-1)` is true when the orientation of `object-1` is straight along x-axis of the reference frame attached to `object-2`.

`StraightAlong-x(object-1)` $\Longleftrightarrow$
`object-type(object-1) = Directional` $\wedge$
`dot-product(unit-vector(object-1, z), unit-vector(object-2, x))` $= 1$.

Definition 6.35: Predicate `StraightAlong-y(object-1)` is true when the orientation of `object-1` is straight along y-axis of the reference frame attached to `object-2`.

`StraightAlong-y(object-1)` $\Longleftrightarrow$
`object-type(object-1) = Directional` $\wedge$
`dot-product(unit-vector(object-1, z), unit-vector(object-2, y))` $= 1$.

Definition 6.36: Predicate `StraightAlong-z(object-1)` is true when the orientation of `object-1` is straight along z-axis of the reference frame attached to `object-2`.

`StraightAlong-z(object-1)` $\Longleftrightarrow$
`object-type(object-1) = Directional` $\wedge$
`dot-product(unit-vector(object-1, z), unit-vector(object-2, z))` $= 1$.

Definition 6.37: Predicate `InvertedAlong-x(object-1)` is true when the orientation of `object-1` is in the direction of negative x-axis of the reference frame attached to `object-2`.

`InvertedAlong-x(object-1)` $\Longleftrightarrow$
`object-type(object-1) = Directional` $\wedge$
`dot-product(unit-vector(object-1, z), unit-vector(object-2, x))` $= -1$.

Definition 6.38: Predicate `InvertedAlong-y(object-1)` is true when the orientation of `object-1` is in the direction of negative y-axis of the reference frame attached to `object-2`.

`InvertedAlong-y(object-1)` $\Longleftrightarrow$
`object-type(object-1) = Directional` $\wedge$
`dot-product(unit-vector(object-1, z), unit-vector(object-2, y))` $= -1$.

Definition 6.39: Predicate `InvertedAlong-z(object-1)` is true when the orientation of `object-1` is in the direction of negative z-axis of the reference frame attached to `object-2`.

`InvertedAlong-z(object-1)` $\Longleftrightarrow$
`object-type(object-1) = Directional` $\wedge$
`dot-product(unit-vector(object-1, z), unit-vector(object-2, z))` $= -1$.

Definition 6.40: Predicate `ParallelTo-xy(object-1)` is true when the orientation of `object-1` is parallel to xy-plane of the reference frame attached to `object-2`.

`ParallelTo-xy(object-1)` $\Longleftrightarrow$
`object-type(object-1) = Directional` $\wedge$
`dot-product(unit-vector(object-1, z), unit-vector(object-2, z))` $= 0 \wedge$
$\neg$`(StraightAlong-x(object-1)` $\vee$ `InvertedAlong-x(object-1))` $\wedge$
$\neg$`(StraightAlong-y(object-1)` $\vee$ `InvertedAlong-y(object-1))`.

Definition 6.41: Predicate `ParallelTo-yz(object-1)` is true when the orientation of `object-1` is parallel to yz-plane of the reference frame attached to `object-2`.

`ParallelTo-yz(object-1)` $\Longleftrightarrow$
`object-type(object-1) = Directional` $\wedge$
`dot-product(unit-vector(object-1, z), unit-vector(object-2, x))` $= 0 \wedge$
$\neg$`(StraightAlong-y(object-1)` $\vee$ `InvertedAlong-y(object-1))` $\wedge$
$\neg$`(StraightAlong-z(object-1)` $\vee$ `InvertedAlong-z(object-1))`.

Definition 6.42: Predicate `ParallelTo-zx(object-1)` is true when the orientation of `object-1` is parallel to zx-plane of the reference frame attached to `object-2`.

`ParallelTo-xy(object-1)` $\Longleftrightarrow$
`object-type(object-1) = Directional` $\wedge$
`dot-product(unit-vector(object-1, z), unit-vector(object-2, y))` $= 0 \wedge$
$\neg$`(StraightAlong-z(object-1)` $\vee$ `InvertedAlong-z(object-1))` $\wedge$
$\neg$`(StraightAlong-x(object-1)` $\vee$ `InvertedAlong-x(object-1))`.

Definition 6.43: Predicate `ArbitraryOrientation(object-1)` is true when `object-1` has an arbitrary orientation in the reference frame attached to `object-2`.

`ArbitraryOrientation(object-1)` $\Longleftrightarrow$
`object-type(object-1) = Directional` $\wedge$
$\neg$`(StraightAlong-x(object-1)` $\vee$ `InvertedAlong-x(object-1))` $\wedge$
$\neg$`(StraightAlong-y(object-1)` $\vee$ `InvertedAlong-y(object-1))` $\wedge$
$\neg$`(StraightAlong-z(object-1)` $\vee$ `InvertedAlong-z(object-1))` $\wedge$
$\neg$`(ParallelTo-xy(object-1)`$\vee$`ParallelTo-yz(object-1)`$\vee$`ParallelTo-zx(object-1))`.

Definition 6.44: Predicate `None(object-1)` is true when `object-1` is a `NonDirectional` object.

`None(object-1)` $\Longleftrightarrow$
`object-type(object-1) = NonDirectional`.

Limits on parameters in Δ

Since the predicates 6.34 - 44 cover the aspect of the orientation of `object-1`, we can use them to estimate limits of *roll* (ρ), *yaw* (θ) and *pitch* (ϕ) of this object. In our settings we define *roll* as the rotation of the object around the x-axis of the frame of reference[5] attached to the geometric center of the object. Similarly *yaw* and *pitch* are defined to be the rotations of the object along y and z axes respectively of the same reference frame. Table 6.4 shows the limits imposed on ρ, θ and ϕ by the orientation of `object-1`. In the table, the values selected for the limits are based on simple principles. That is, along any axis we allow the maximum rotation of the object to be within $\frac{\pi}{2}$ radians (i.e. $(-\pi/4, \pi/4)$). We restrict the rotation along any axis to $\frac{\pi}{8}$ radians (i.e. $(-\pi/16, \pi/16)$) if the truth value of a predicate (in the final state of the object) is more sensitive to the rotation of the object around that axis (in the initial state of the object). We do not put any limits on the values of the parameters if the truth value of a predicate is independent of the rotation along a particular axis. We also do not put any limits on the parameters for the cases where the object has `ArbitraryOrientation` or it is `NonDirectional`. In our implementation we replace the ∞ symbol with $\frac{\pi}{4}$.

Predicates	$Lim_{P2}\rho$	$Lim_{P2}\theta$	$Lim_{P2}\phi$
`StraightAlong-x`	$(-\infty, \infty)$	$(-\pi/4, \pi/4)$	$(-\pi/16, \pi/16)$
`StraightAlong-y`	$(-\pi/4, \pi/4)$	$(-\infty, \infty)$	$(-\pi/16, \pi/16)$
`StraightAlong-z`	$(-\pi/16, \pi/16)$	$(-\pi/16, \pi/16)$	$(-\infty, \infty)$
`InvertedAlong-x`	$(-\infty, \infty)$	$(-\pi/4, \pi/4)$	$(-\pi/16, \pi/16)$
`InvertedAlong-y`	$(-\pi/4, \pi/4)$	$(-\infty, \infty)$	$(-\pi/16, \pi/16)$
`InvertedAlong-z`	$(-\pi/16, \pi/16)$	$(-\pi/16, \pi/16)$	$(-\infty, \infty)$
`ParallelTo-xy`	$(-\pi/4, \pi/4)$	$(-\pi/4, \pi/4)$	$(-\infty, \infty)$
`ParallelTo-yz`	$(-\infty, \infty)$	$(-\pi/4, \pi/4)$	$(-\pi/4, \pi/4)$
`ParallelTo-zx`	$(-\pi/4, \pi/4)$	$(-\infty, \infty)$	$(-\pi/4, \pi/4)$
`ArbitraryOrient`	$(-\infty, \infty)$	$(-\infty, \infty)$	$(-\infty, \infty)$
`None`	$(-\infty, \infty)$	$(-\infty, \infty)$	$(-\infty, \infty)$

Table 6.4: Limits imposed on ρ, θ and ϕ by the orientation of `object-1`.

6.5 Motion

In this section we give the definitions of the predicates in the description vocabulary which are related to the aspect of motions of the objects. We consider linear and angular

[5]This frame can be seen as a translated version of the frame of reference attached to `object-2`. Since translation of `object-1` is independent of its orientation, in chapter 7 we treat both the reference frames as a single frame of reference.

motion of the objects in this vocabulary. Since in our settings we consider `object-2` to be static, therefore the domain of the variables of the unary predicates of motion vocabulary is restricted to `object-1`.

6.5.1 Linear

In a three dimensional space linear velocity of a moving object has three components. These components are represented as $\dot{x}, \dot{y}$ and $\dot{z}$ in this work. In these components $\dot{x}$ is the component of the velocity along x-axis of the frame of reference attached to `object-2`. Similarly, $\dot{y}$ and $\dot{z}$ are the components along y and z axes respectively of the same frame of reference. We consider a linear velocity components to be positive if the direction of the component is along the direction of increasing values of the coordinate of the axis. Otherwise, the component is considered negative. Based on this description we can formulate following definitions:

Description vocabulary for linear motion of `object-1`

Definition 6.45: Predicate `Moving-PosVx(object-1)` is true when the x-component of the linear velocity of `object-1` (i.e. $\dot{x}$) is greater than zero.

`Moving-PosVx(object-1)` $\iff \dot{x} > 0$.

Definition 6.46: Predicate `Moving-NegVx(object-1)` is true when the x-component of the linear velocity of `object-1` (i.e. $\dot{x}$) is smaller than zero.

`Moving-NegVx(object-1)` $\iff \dot{x} < 0$.

Definition 6.47: Predicate `Stationary-Vx(object-1)` is true when the x-component of the linear velocity of `object-1` (i.e. $\dot{x}$) is equal to zero.

`Stationary-Vx(object-1)` $\iff \dot{x} = 0$.

The above definitions of the predicates are very straightforward. We also create similar definitions for $\dot{y}$ and $\dot{z}$. Here, we do not present these definitions explicitly to avoid stating the obvious. However, it is worth mentioning that in the description vocabulary the aspect of linear motion is captured completely with the help of conjunction of these definitions (i.e. the predicates). That is, `P4` (in table 5.2) is actually represented by a conjunction of predicates. For example, if we need to represent the fact that a `object-1` is moving in the direction of positive x-axis of the frame of reference attached to `object-2`, then:

`P4` $\equiv$
`Moving-PosVx(object-1)` $\wedge$ `Stationary-Vy(object-1)` $\wedge$ `Stationary-Vz(object-1)`.

To express the fact that an object is completely stationary we make use of a predicate `Stationary-V/1`, which is defined in definition 6.54. In case this predicate is true, then `P4` $\equiv$ `Stationary-V(object-1)`.

Definition 6.54[6]: Predicate `Stationary-V(object-1)` is true when all the components of the linear velocity of `object-1` are equal to zero.

`Stationary-V(object-1)` $\iff \dot{x} = \dot{y} = \dot{z} = 0$.

Limits on parameters in Δ

We use simple rules for putting limits on the parameters related to the linear motion of the object. These rules are as following:

For $\delta \in [\dot{x}, \dot{y}, \dot{z}]$;

- If $\delta > 0$, then the interval of the limits of δ is [sv, m].
- If $\delta < 0$, then the interval of the limits of δ is [-m, -sv].
- If $\delta = 0$, then the interval of the limits of δ is [-sv, sv].

In above mentioned limits, 'sv' (in m/s) represents the smallest non-zero positive real number that can be considered for velocity in the simulation process, whereas 'm' is the maximum value of the velocity that can be transmitted to an object by a robot. In our experiments (presented in chapter 7), we choose sv = 0.001 m/s and m = 0.75 m/s.

6.5.2 Angular

Similar to the linear motion of an object, its angular motion can also be described by three orthogonal components in a three dimensional space. These components are represented by $\dot{\rho}, \dot{\theta}$ and $\dot{\phi}$ in our settings. These components are simply the rate of change of ρ, θ and ϕ respectively. Since along any axis an object can rotate either in clock wise (CW) or counter clock wise (CCW) direction, we can formulate following definitions to describe its rotation:

Description vocabulary for angular motion of `object-1`

Definition 6.55: Predicate `Rotating-CCWx(object-1)` is true when an `object-1` is rotating in CCW direction along the x-axis of the frame of reference attached to `object-2`.

`Rotating-CCWx(object-1)` $\iff \dot{\rho} > 0$.

Definition 6.56: Predicate `Rotating-CWx(object-1)` is true when an `object-1` is rotating in CW direction along the x-axis of the frame of reference attached to `object-2`.

[6]Index number 6.48 - 53 are reserved for the predicates related to $\dot{y}$ and $\dot{z}$

`Rotating-CWx(object-1)` $\Longleftrightarrow \dot{\rho} < 0$.

Definition 6.57: Predicate `Rot-StationaryX(object-1)` is true when an `object-1` is not rotating in any direction along the x-axis of the frame of reference attached to `object-2`.

`Rot-StationaryX(object-1)` $\Longleftrightarrow \dot{\rho} = 0$.

We also create similar definitions with the help of $\dot{\theta}$ and $\dot{\phi}$. However, we do not show these definitions here explicitly. Similar to the case of linear motion, the aspect of angular motion is captured completely with the help of conjunction of the definitions (i.e. predicates). Therefore for an `object-1` rotating CW around the x-axis of the of the frame of reference attached to `object-2`, the predicate `P5` (in table 5.2) can be represented as following:

`P5` $\equiv$
`Rotating-CWx(object-1)` $\wedge$ `Rot-StationaryY(object-1)` $\wedge$
`Rot-StationaryZ(object-1)`.

To describe the condition when an `object-1` is not rotating around any axis, we define the predicate `Stationary-AV(object-1)`. Definition 6.64 describes this predicate. If `Stationary-AV/1` is true then:

`P5` $\equiv$ `Stationary-AV(object-1)`.

Definition 6.64[7]: Predicate `Stationary-AV(object-1)` is true when all the components of the angular velocity of `object-1` are equal to zero.

`Stationary-AV(object-1)` $\Longleftrightarrow \dot{\rho} = \dot{\theta} = \dot{\phi} = 0$.

Limits on parameters in Δ

Similar to the case of linear motion, we find the limits of the parameters using following rules:

For $\delta \in [\dot{\rho}, \dot{\theta}, \dot{\phi}]$;

- If $\delta > 0$, then the interval of the limits of δ is [sa, ma].
- If $\delta < 0$, then the interval of the limits of δ is [-ma, -sa].
- If $\delta = 0$, then the interval of the limits of δ is [-sa, sa].

In above mentioned limits, 'sa' (in rad/s) represents the smallest non-zero positive real number that can be considered for angular velocity in the simulation process, whereas 'ma' is the maximum value of the angular velocity that can be transmitted to an object by a robot. In our experiments (presented in chapter 7), we choose sa = 0.001 rad/s and ma = 0.75 rad/s.

[7]Index number 6.58 - 63 are reserved for the predicates related to $\dot{\theta}$ and $\dot{\phi}$

6.6 Containment

In this section we define three more predicates that describe the aspect of *containment* of the objects. These predicates can be used for the actions when `object-2` is a container that can contain `object-1` inside it. In our settings the aspect of containment is analogous to the aspect of connectedness of the objects. This means, in our work when `object-2` is a container then the predicate `P1` (in table 5.2) represents the aspect of containment of the objects instead of their connectedness, and `P1` is one of the three predicates given below. The predicates defined for the aspect of containment make use of `Over/2` predicate and `height/1` function from section 6.2. The other function used in the definition is given below.

Definition 6.6.1: Function `highest(pp)` refers to the highest point in the set of points `pp`. The highest point is the one with the largest z-coordinate value. The points in `pp` are in $\mathbf{R}^3$.

Description vocabulary for the aspect of containment of objects

Definition 6.65: Predicate `In(object-1, object-2)` is true when `object-1` is contained completely inside `object-2`.

`In(object-1, object-2)` $\iff$
`Over(object-1, object-2)` $\wedge$
`(highest(object-1) < height(object-2))`.

Definition 6.66: Predicate `Partially-In(object-1, object-2)` is true when `object-1` is contained partially inside `object-2`.

`Partially-In(object-1, object-2)` $\iff$
`Over(object-1, object-2)` $\wedge$
`(highest(object-1) > height(object-2))` $\wedge$
`(lowest(object-1) < height(object-2)`.

Definition 6.67: Predicate `Out(object-1, object-2)` is true when `object-1` is outside `object-2`.

`Out(object-1, object-2)` $\iff$
$\neg$`In(object-1, object-2)` $\wedge$
$\neg$`Partially-In(object-1, object-2)`

Limits on parameters in Δ

Similar to the case of *connectedness* predicates, the predicates related to the aspect of *containment* do not place any limits on the values of x and y. For the values of z we use exactly the same rules for the limits as those used in the case of *connectedness* in section 6.2. The only difference in the implementation of these rules is that here we use the top surface of the bottom of the container (i.e. `object-2`) to make measurements instead of using the highest surface of `object-2`.

7 Results and analysis

In this chapter we present the results of application of our approach to the action of dropping objects over other objects. The results are given in the form of different experiments and description of each experiment is organized according to the *steps* of the schematics (shown in figure 5.2) of our approach. We explain the results of initial experiments in detail and gradually reduce the descriptive explanation of the results in the latter experiments to avoid repetition.

In the experiments presented below, `object-1` is always chosen to be an object that can be easily manipulated by a manipulator (e.g. `die`, `bottle`). The domain of `object-2` consists of *static* objects which can either be used to support `object-1` (e.g. `table`) or contain `object-1` (e.g. `basket`). In the results, the measurements of lengths and coordinates are given in *meters*, whereas the measurements of angles are given in *radians*. Similarly, linear velocities are measured in *meters per second* and angular velocities are measured in *radians per second*. The measurements of the coordinates are according to the reference frame that has its origin on the floor, exactly below the geometric center of the top surface of `object-2`, as shown in the figure on the right. In this figure, '`h`' represents the height of `object-2` and ρ, θ and ϕ correspond to *roll, yaw* and *pitch* measurements receptively. For these measurements angles are measured positive for counter clock wise (CCW) directions, when seen into the plane of rotation from higher values of the axis of rotation. Components of the angular velocities are also assumed to be positive for CCW rotations in a similar manner. Components of linear velocities are measured positive when their directions are same as the directions of respective positive axes of the references frame.

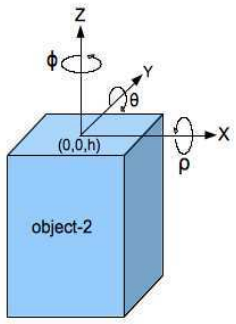


Figure 7.1: Frame of reference illustration.

In the explanation of the results we use the terms *parameters* and *features* interchangeably depending upon the context. Both these terms refer to the parameters in Δ. We also use the phrases, *process of example generation* and *simulation process* in a similar manner. These phrases refer to the execution of the algorithm shown in figure 5.9. All the terms, phrases and symbols used in this chapter refer to the same concepts as they do in previous chapters.

7.1 Experiment 1

Specifications

In this experiment we use a `die` as `object-1` and a regular `table` with square top, as `object-2`. Specifications of these objects are shown in tabular form below. In the dimensions of the objects, height (h) is measured along z-axis, length (l) is measured along y-axis and width (w) is measured along x-axis of the frame of reference mentioned in figure 7.1. Using these specification we create a simulation that simulates an action in

Objects	**Dimensions** (m)	**Mass** (kg)
`Die`	0.14(h) × 0.14(l) × 0.14(w)	0.8
`Table`	0.74(h) × 0.6(l) × 0.6(w)	∞[1]

Table 7.1: Specifications of the objects for experiment 1.

which the `die` is dropped `over` the center of the top surface of the `table`. The `die` stays `on` the `table` in the final state of the simulation. In the initial state of the simulation all parameters in Δ have values equal to 0.0, except z. For z, the value is 1.0. The simulation shows the *sample behavior* of the objects, which is expected if the action of releasing the `die` on the `table` is performed successfully.

Simulation description

Using the information from the simulation of the sample behavior (i.e. object models, Δ and $\boldsymbol{\mu}$) , *step 1* of the approach finds the description of the simulation. In this experiment *step 1* places 588 markers on the `die` and 4 markers on the top surface of the `table`. With the help of these markers following expression in FOL is extracted to describe the initial state of the simulation:

$\mathtt{S}_{init} \equiv$ `Over(Die, Table)` $\wedge$ `Zz(Die, Table)` $\wedge$ `None(Die)`$\wedge$ `Stationary-V(Die)` $\wedge$ `Stationary-AV(Die)`.

Once $\mathtt{S}_{init}$ is found, *step 1* simulates the objects to find the description of the final state of the objects. Following is the description found for the final state of the objects in this experiment:

$\mathtt{S}_{final} \equiv$ `On(Die, Table)` $\wedge$ `Zz(Die, Table)` $\wedge$ `None(Die)`$\wedge$ `Stationary-V(Die)` $\wedge$ `Stationary-AV(Die)`.

In our approach, we consider the final state of the objects as a state in which both[1] the objects become stationary. We terminate the simulation when this state is achieved. Although it is not mentioned earlier in the description of the approach, but we also take

[1]In our experiments `object-2` is always static.

advantage of the time of the sample simulation. That is, we execute the simulation of the sample behavior three times and store the average time taken by the simulation to reach its final state. This information is later utilized to terminate the simulation (in *step 2*) in the cases where the behavior of the objects become undesirable because the simulation takes too long to achieve the final state. In this experiment, the average time of the simulation is 1.13 *sec*.

Limits of the parameters

When $\mathtt{S}_{init}$ and $\mathtt{S}_{final}$ are known, *step 2* of the approach finds an initial estimate of the limits of the values for each parameter δ in Δ using only $\mathtt{S}_{init}$. With these limits, *step 2* executes the algorithm shown in figure 5.9 to generate 5000 instances of the action. During the execution of the algorithm, the limits get refined to new values. Both the initial and the refined values of the limits of each parameter for this experiment are given in table 7.2.

Parameter(s) δ(s)	**Lower limit** Initial	Refined	**Upper limit** Initial	Refined
x	-0.3750	-0.2131	0.3750	0.2101
y	-0.3750	-0.2139	0.3750	0.2153
z	0.8100	0.8168	1.0000	0.9998
ρ	-0.7853	-0.7841	0.7853	0.7810
θ	-0.7853	-0.7821	0.7853	0.7841
ϕ	-0.7853	-0.7848	0.7853	0.7842
$\dot{x}, \dot{y}, \dot{z}, \dot{\rho}, \dot{\theta}, \dot{\phi}$	-0.0010	-0.0009	0.0010	0.0009

Table 7.2: Initial and refined limits of the parameters in Δ, for experiment 1.

In the algorithm of figure 5.9, the change in the values of limits occurs when the process of example generation meets the conditions given in line '3' of the algorithm. We refer to this point in the execution of the algorithm as the *transition* stage of the simulation process, and the point of termination of the algorithm is referred as the *termination* stage of the simulation process. Table 7.3 gives the information regarding the number of negative and positive examples generated at the aforementioned stages of the simulation process. The

Stage of simulation process	+ve examples	−ve examples	Extra evaluations
Transition	776	76	2923
Termination	4618	382	4524

Table 7.3: States of the simulation process for experiment 1.

last column of this table gives the number of *extra evaluations* of $\mathtt{S}_{init}$ in the simulation process. These evaluations correspond to the initial states of the objects when the values of the parameters are selected from the estimated limits but the `P1` predicate corresponding

to these values is not the same as the P1 predicate in the sample simulation (see line '12' in figure 5.9). The table shows that in the initial 852 (i.e. $776 + 76$) simulations a total of 2923 extra evaluations were conducted, but after the refinement of limits only 1601 (i.e. $4524 - 2923$) extra evaluations were conducted in 4148 (i.e. $5000 - 852$) simulations. This significant change in the proportion of extra evaluations is because of the fact that after transition stage of the simulation process we use refined limits to generate the initial states of the objects. Since each evaluation of $\mathtt{S}_{init}$ is done using 592 (i.e. $588+4$) markers on the objects, avoiding each evaluation causes a significant computational advantage. It should be noticed that if we do not use the refined limits after the transition stage of the simulation process then we need to conduct approximately 17154 extra evaluations for 5000 instances. This shows one advantage of dynamically refining the limits of parameters and using the refined limits in the simulation process.

We can make following two observations regarding the values given in table 7.3:

- The percentages of the negative and the positive examples are almost similar at both the stages of the simulation process. That is, in the initial 852 examples 8.9% examples are negative and 91.1% examples are positive, and in 5000 examples 7.6% examples are negative and 92.4% positive. This shows that changing the limits in the simulation process does not affect the distributions of the examples significantly.
- In this experiment, percentage of negative examples in 5000 examples shows that a robot can already perform the action very reliably if it chooses the values of parameters from the refined limits shown in table 7.2. Using maximum likelihood we can say that (for this experiment) given the values of the parameters are chosen within the refined limits, the probability of performing the action successfully is 0.924. Let us denote this probability as $P(success|refinedLimits)$. So, for this experiment:

$$P(success|refinedLimits) = 0.924$$

In this experiments and the experiments to follow, we also generate the examples by allowing the algorithm in figure 5.9 to execute until there are at least 2500 examples for both of the types of examples (i.e. positive and negative). From these examples we choose 5000 examples which comprise 2500 positive and 2500 negative examples. The reasons for selecting the data with equal number of positive and negative examples (i.e. *uniform data*) will become apparent shortly. Here, we like to highlight the fact that for this experiment 2500 negative examples were generated after 26271 simulations and the time required by the complete simulation process[1] to generate these examples was 52.74

[1]The simulation was run on a machine with 2.53 GHz Intel Core 2 Duo processor with 4 GB memory using Windows XP. These specifications are same for all the experiments reported here.

hrs (i.e. 189871 *sec*). Let us denote this time by $TSim_u$. In contrast to $TSim_u$, the time required for the 5000 (i.e. $4618 + 382$) examples originally generated by the algorithm of figure 5.9 is 6.33 *hrs* (i.e. 22796 *sec*). Let us denote this time by $TSim_s$. So, for this experiment

$$TSim_u = 8.33 \times TSim_s$$

In the notations $TSim_u$ and $TSim_s$, subscript u denotes *uniform data* and s denotes *skewed data*. In ML literature, *uniform data* and *skewed data* are commonly used terms for the types of data discussed here.

Modification of planning operator

For this experiment when we separately group the values of each parameter or *feature* of the data according to the labels of the *classes* of the examples, then we get the plots shown in figure 7.2, 7.3 and 7.4[2]. These plots are for uniform data with 5000 instances. In the plots, labels *Class 0* and *Class 1* correspond to negative examples and positive examples respectively. The black and green lines in the plots correspond to the limits of the bins of the minimum and the maximum *BinStrength*s repectively. If we concentrate only on the distributions of the values of the parameters, then we can see some patterns in the distributions of the values of x, y, z, ρ and θ. Whereas, the distributions for $\phi, \dot{x}, \dot{y}$ and $\dot{z}$ (and $\dot{\rho}, \dot{\theta}, \dot{\phi}$) are almost completely random. This notion is also reflected in the weights of the parameters in figure 7.5(a). The weights in the figure emphasize the fact that, given that the values of the parameters are chosen within the refined limits (given in table 7.2), it is still important to select the values for x, y, z, ρ and θ carefully. This careful selection is evident in figure 7.2 and 7.3 in the form of locations and widths of the bins. The N-Bins algorithm (shown in figure 5.15) suggests the values of the parameters as the mean values of the limits of the bins shown in green color. It is visible from the figures that for the parameters with high weights in figure 7.5(a), the green bins have less widths and the bins are chosen where concentrations of the blue points are much more than the those of red points. For the unimportant parameters (e.g. $\phi, \dot{x}$) the green bins are chosen almost randomly.

[2]We do not show the plots of $\dot{\rho}, \dot{\theta}, \dot{\phi}$ because distributions of the values for these parameters are almost the same as the distributions of the values of $\dot{x}, \dot{y}, \dot{z}$.

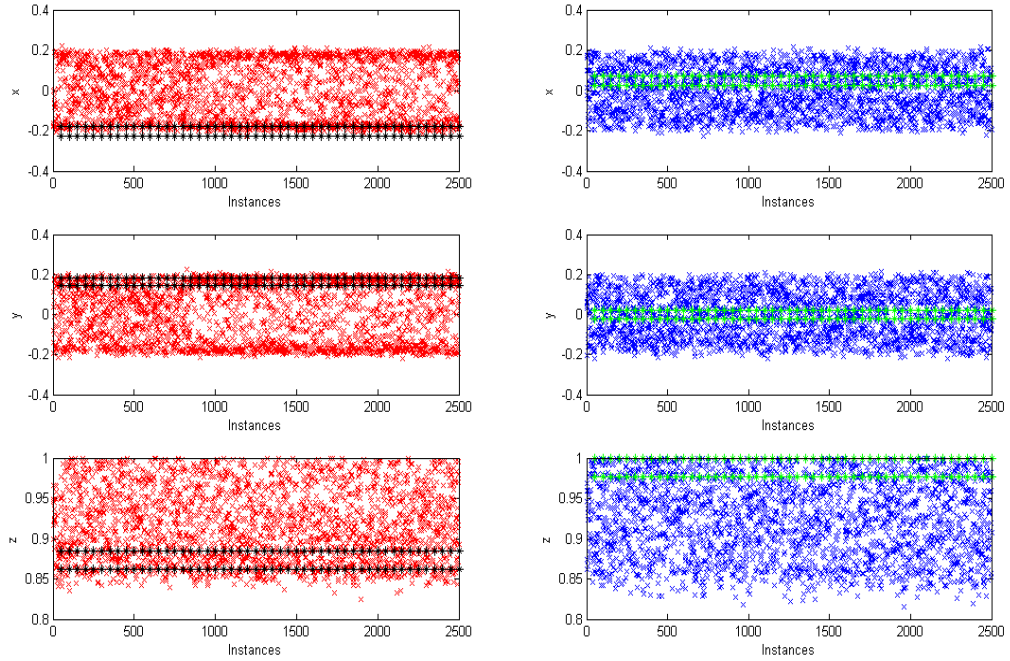


Figure 7.2: Distribution of x, y, z according to class labels, using uniform data (5000 instances).

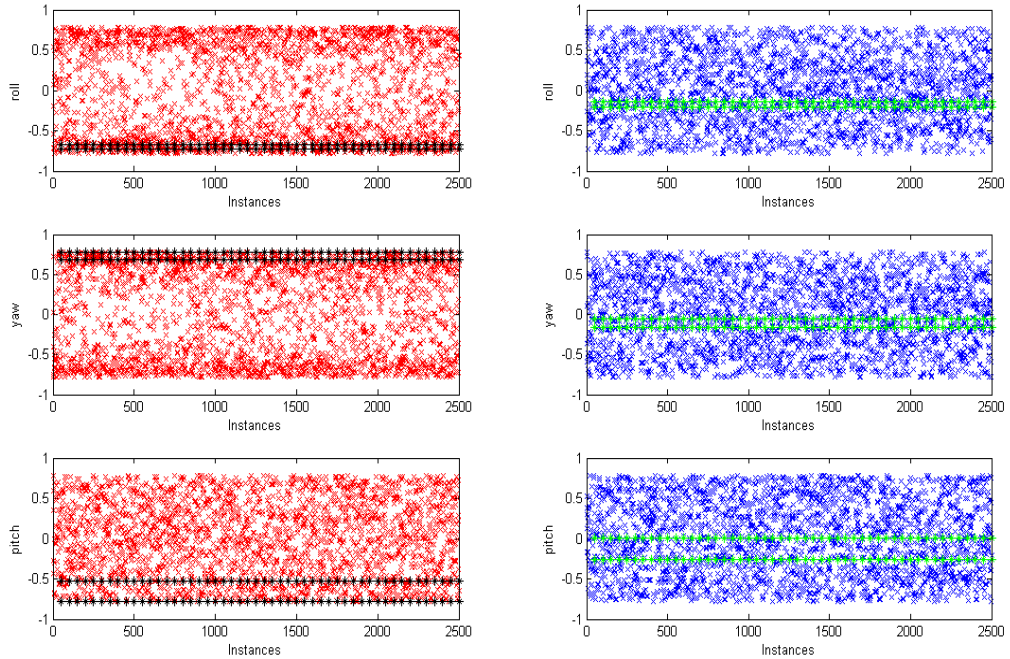


Figure 7.3: Distribution of ρ, θ, ϕ according to class labels, using uniform data (5000 instances).

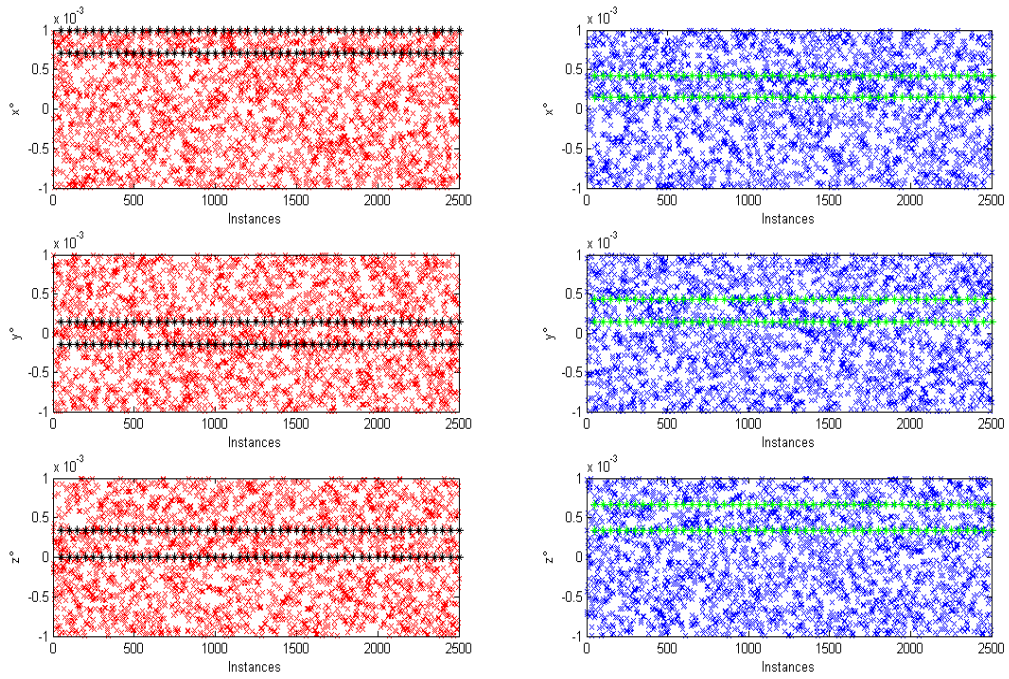
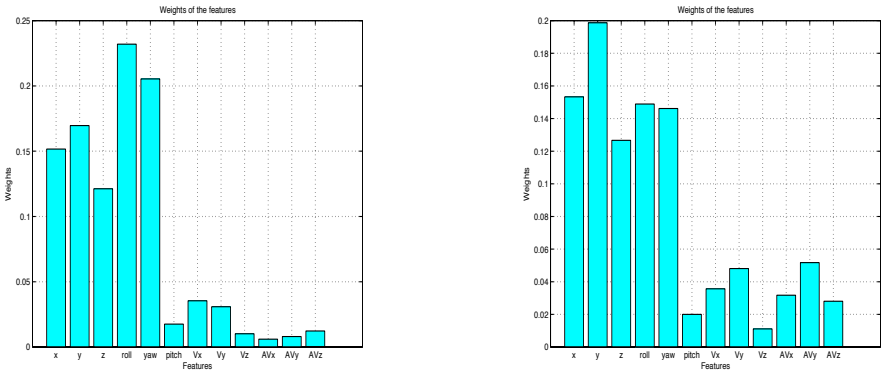


Figure 7.4: Distribution of $\dot{x}, \dot{y}, \dot{z}$ according to class labels, using uniform data (5000 instances).



(a) 5000 instances (uniform data) (b) 3000 instances (skewed data)

Figure 7.5: Comparison of the weights of the parameters for two different sets of instances.

Table 7.4 shows the values of the parameters[3] suggested by N-Bins algorithm to reliably perform the action of releasing the `die` over the `table`. The table also shows the limits of the bins from which the suggested values are calculated (as the mean values of the limits). The last column of the table shows the number of bins in which N-Bins divides the refined limits of each parameter to calculate the suggested values. All the values shown in the table use $binMul = 2$ and $augBin = 3$ in figure 5.13. Both these values are constant for all the experiments reported in this chapter.

Parameter(s) δ(s)	**Suggested values**	**Bin limits** Lower	Upper	**Number of bins**
x	0.0470	0.0222	0.0719	9
y	0.0012	-0.0190	0.0214	11
z	0.9885	0.9769	1.0000	8
ρ	-0.1744	-0.2035	-0.1453	27
θ	-0.1050	-0.1573	-0.0526	15
ϕ	-0.1309	-0.2618	0.0000	6

Table 7.4: Selected values and bins of the parameters based on uniform data (5000 instances) for experiment 1.

We refer to the data generated by the regular execution of the simulation process as the *skewed data* (i.e. the data represented by table 7.3). From the 5000 instances of the skewed data, we select first 3000 instances and find weights of the parameters based on these instances. Figure 7.5(b) shows these weights. In figure 7.6 we also give the plots of the distributions of the values of x, y and z for the same instances. If we compare these figures from the corresponding figures of 5000 instances of uniform data, we can notice two important observations:

1. Although the exact values of the weights of corresponding parameters in figure 7.5 are different, but the important features in figure (a) also have high weights in figure (b). This implies that with respect to 5000 instances of the uniform data, the characteristic of relative importance of the parameters is roughly maintained in 3000 instances of the skewed data.

2. Although the bins shown in figure 7.6 are not at the exact same locations as those in figure 7.2, but the locations of the bins in figure 7.6 are in agreement with the distributions of figure 7.2. That is, the green boundaries in figure 7.6 correspond to the areas in 7.2 that have high *binStrengths* and the black boundaries correspond to the regions with very low *binStrengths*. Furthermore, the widths of the corresponding bins in both figures are also similar.

[3]The parameters do not include $\dot{x}, \dot{y}, \dot{z}, \dot{\rho}, \dot{\theta}, \dot{\phi}$ because these parameters are not important in this experiment. In the experiments to follow, we also skip the information regarding other unimportant parameters.

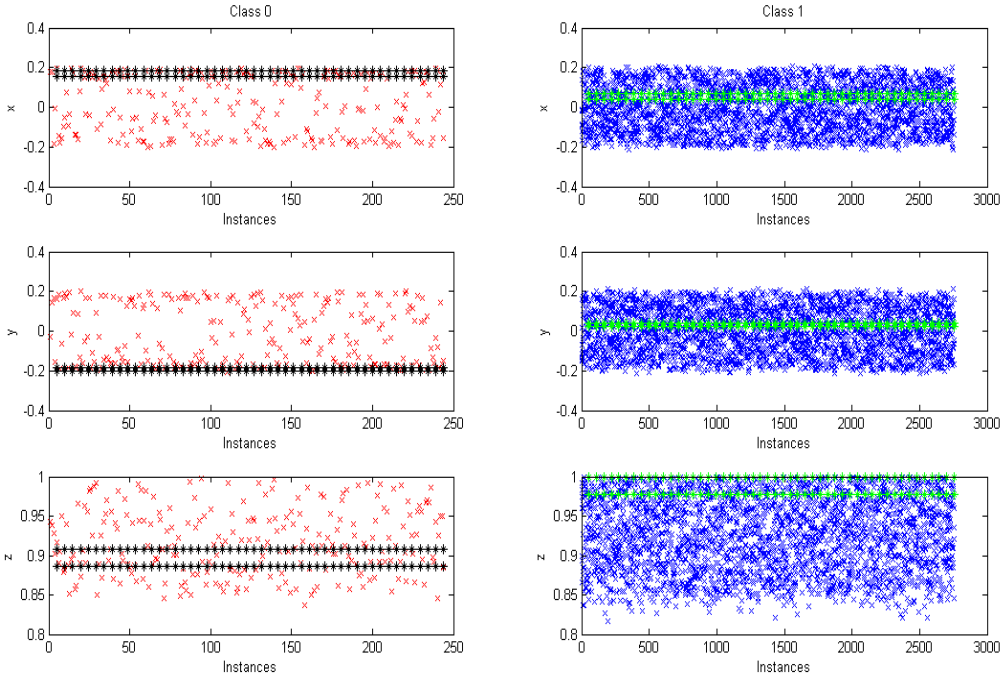


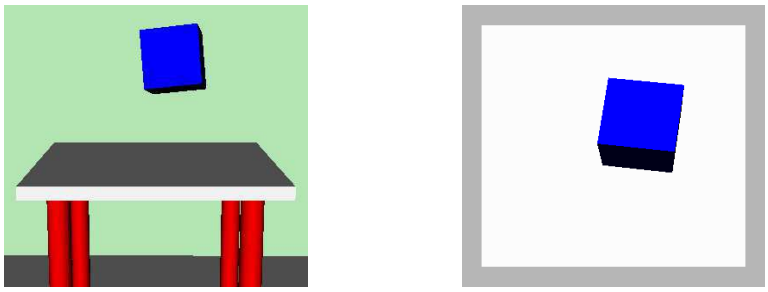
Figure 7.6: Distribution of x, y, z according to class labels, using skewed data (3000 instances).

For the skewed data we show the distributions of only x, y, z parameters in figure 7.6. Observation (2) is true in general for all the parameters. Table 7.5 shows the suggested values of the parameters and, limits and number of bins for the given parameters for skewed data (3000 instance).

Parameter(s) δ(s)	**Suggested values**	**Bin limits** Lower	Upper	**Number of bins**
x	0.0547	0.0406	0.0689	15
y	0.0321	0.0241	0.0400	27
z	0.9885	0.9771	0.9999	8
ρ	-0.0012	-0.0723	0.0700	11
θ	0.0012	-0.0858	0.0882	9
ϕ	-0.1302	-0.0006	0.2610	6

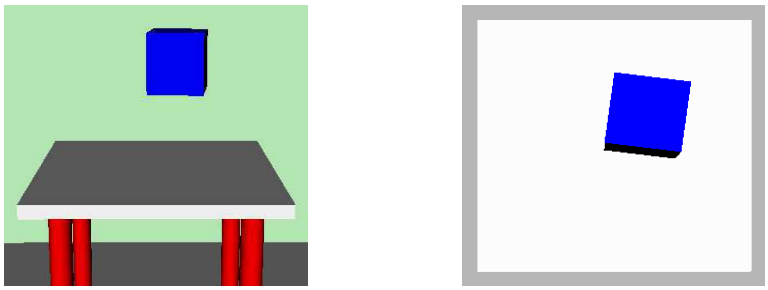
Table 7.5: Selected values and bins of the parameters based on skewed data (3000 instances) for experiment 1.

When we use the values of the parameters given (as 'suggested values') in table 7.4 in simulation then the initial state of the objects in the simulation look like figure 7.7. This figure shows the views of the objects in the initial state suggested by the N-Bins algorithm when the algorithm suggests the values of the parameters by using uniform data. Similarly, figure 7.8 shows the views of the suggested initial state of the objects



(a) Front view (b) Top view

Figure 7.7: Suggested initial state for experiment 1, based on uniform data (5000 instances).



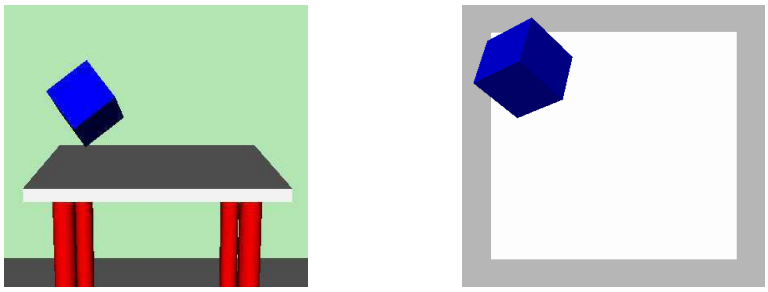
(a) Front view (b) Top view

Figure 7.8: Suggested initial state for experiment 1, based on skewed data (3000 instances).

based on the skewed data (3000 instances). Values of the parameters in this state are taken from table 7.5.

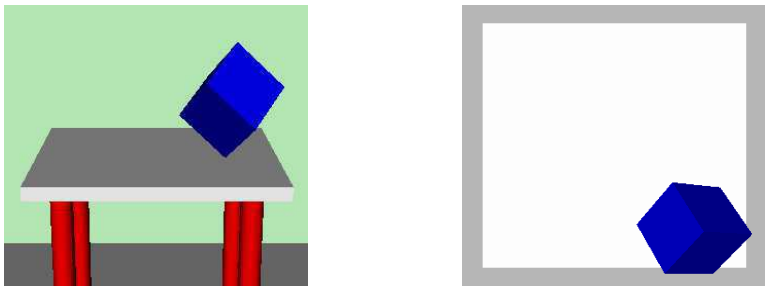
Similar to the suggested initial state, we can also estimate the worst possible initial state of the objects by using the values from the bins shown in black lines in the plots of distributions of the values of the parameters. Figures 7.9 and 7.10 show the estimated worst initial states for the objects, calculated by using uniform and skewed data respectively in the N-Bins algorithm. It should be noticed that these are the worst initial states to release the `die`, given all the parameters of the objects are chosen within the refined limits. In other words, these are potentially the situations which can cause external faults despite the fact that values of the parameters are selected within the refined limits. Existence of these situations implies that it is not enough to rely only on the refined limits of the parameters to avoid the occurrence of external faults. That is why in our approach we go on to find the best bins and exact values of the parameters to suggest the initial state of the objects to perform an action reliably.

For this experiment we also generate a uniform data (with 5000 instances) in which we do not refine the limits in the simulation process. Instead, we run the complete simulation process with the limits of x and y parameters equal to the extents of the top surface of the table. That is $-0.3 \leq x \leq 0.3$ and $-0.3 \leq y \leq 0.3$. Limits of all other parameters are kept same as the initial limits shown in table 7.2. If we calculate the weights of the



(a) Front view (b) Top view

Figure 7.9: Estimated worst initial state for experiment 1, using uniform data (5000 instances).



(a) Front view (b) Top view

Figure 7.10: Estimated worst initial state for experiment 1, using skewed data (3000 instances).

features based on this data we get the weights shown in figure 7.11. It is clear from the figure that in this case x, y and z are the important parameters. Whereas, ρ, θ, ϕ and rest of the parameters do not play any major role in the success of the action. Apparently, this notion is very intuitive. However from the weights of the parameters in figure 7.5(a) and (b), we know that this notion is not true if we select the values of x and y within the refined limits given in table 7.2. Within those limits ρ and θ also become significantly important parameters. The reason for the differences in the weights of the parameters can be understood by the explanation in the next paragraph.

In the case of refined limits all the values of x and y parameters are selected (within the limits) such that the *complete* `die` is over the `table`'s top surface in the initial state. In this case the only way the `die` can fall down from the `table` is that it lands *significantly* tilted on the top surface of the `table` near the corner (while remaining completely over it) and then roll at its own and fall on the floor. Hence, the initial state of the `die` that results in a fault will always be consisting of the values of roll (i.e. ρ) and yaw (i.e. θ) that cause a significant tilt to the `die`. Such values of ρ and θ are strongly related to negative examples. This makes ρ and θ prominent parameters in the simulation process. Whereas in the case of figure 7.11, the `die` can also fall down on the floor even with a little tilt (in addition to the case of significant tilt) when it is released near the corner because its center of mass can be too near to the corner. In other words, the values

of ρ and θ are not very important because no matter what values of these parameters are selected the `die` can fall down on the floor because of the values of x, y parameters.

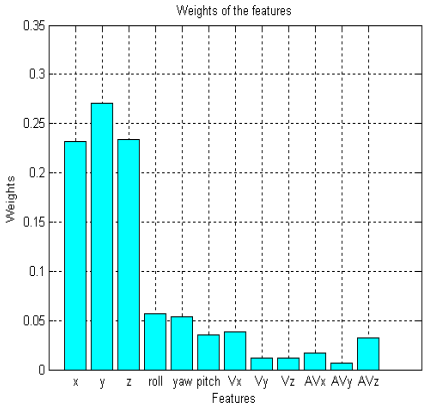


Figure 7.11: Weights of the parameters with limits of x and y equal to the extents of the table.

The explanation above illustrates that how our approach correctly amplifies the importance of the parameters which may look unimportant at first sight. It should be noticed that by avoiding the superficiality in the weights of the parameters, our approach is able to suggest the initial states of the objects that can avoid the occurrence of the *unknown external faults*. As mentioned in section 5.2.4, the N-Bins algorithm can also be used as a binary classification algorithm. In our experiments, we also test the performance (i.e. accuracy) of N-Bins as a binary classifier and compare it with the performances of other state of the art learning algorithms. The reason for this exercise is *condition 3* for *modifying the planning operator* in section 5.2.4. We use following learning algorithms in our experiments:

- Artificial Neural Networks (NN)
- Support Vector Machines (SVM)
- k-Nearest Neighbors (k-NN)
- Decision Trees (DT)

We use RapidMiner[4] for the performance analysis. RapidMiner is an open-source system for data mining that is available as a standalone application. With the help of this software we use the process of cross validation to find the best parameters of the learning algorithms for our experiments. In this process we use 20% instances as the cross validation set of data. Parameters of each learning algorithm obtained after the process of cross validation are given in *appendix (D)*. We conducted thorough experiments to find these parameters,

[4]http://www.rapidminer.com

however we do not provide any further discussion on this issue because these experiments are not the main topic of interest in the thesis. For the same reasons, we also do not provide the details on the hypotheses learnt by the above mentioned learning algorithms. Working principles and hypotheses representations of these algorithms can be found in any ML literature (e.g. Mitchell [1997]).

The questions of interest in this work are that which of the ML algorithms have better accuracies in predicting the final state of the objects when they are provided with the instances of the initial states, and how the accuracies respond to the number of training instances. Furthermore, we are interested in knowing the effects of data distribution (i.e. uniform or skewed) on the accuracies. This knowledge is important because in our experiments we find that generally $TSim_u > TSims$. Therefore, an algorithm performing better on skewed data is preferable when time is considered important. Since we are dealing in robotics, we are also interested in the response of the algorithms for noisy data. In this experiment and others to follow, these topics are the focus of our interest whenever learning algorithms are discussed.

In the results reported in this thesis, we analyze the accuracies of the algorithms by using different numbers of training instances. We give only the plots of these accuracies in this chapter. For each experiment, the accuracies and learning time information is provided in *appendix (E)* of the thesis. We also give the precisions and recalls (see equations 2.1 and 2.2) of the algorithms in this appendix. All the values of accuracies are based on separate 1000 test instances, in which 500 instances belong to each of the classes of instances (i.e. class 0 and class 1). In the graphs of accuracies, numbers of training instances are shown as the x-axes of the plots, whereas the y-axes show the percentage accuracies of the algorithms. For each plot, '$-U$' symbol in the legend denotes uniform training data, whereas '$-S$' symbol denotes skewed training data.

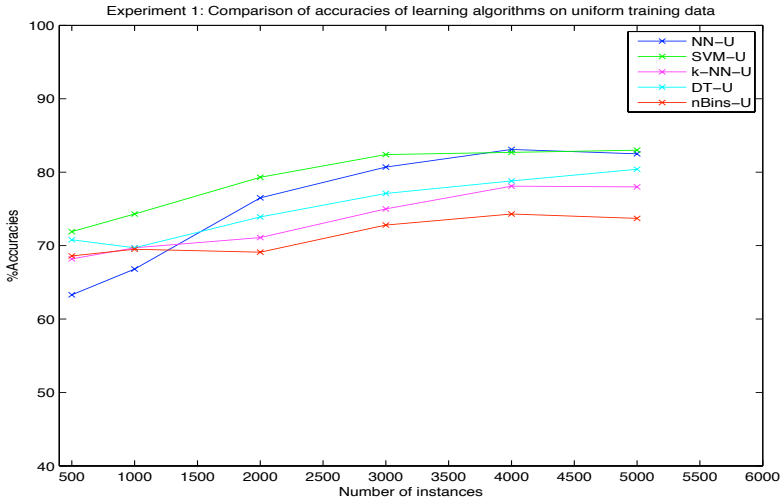


Figure 7.12: Comparison of accuracies of learning algorithms for experiment 1, with uniform training data.

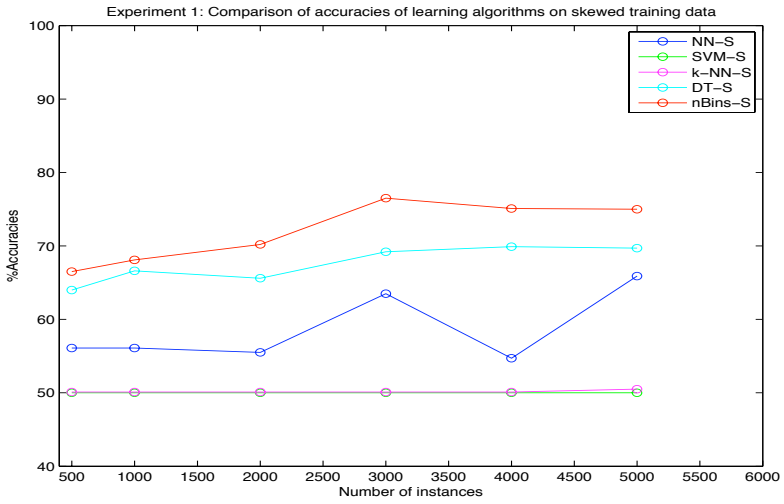


Figure 7.13: Comparison of accuracies of learning algorithms for experiment 1, with skewed training data.

Figure 7.12 shows the plot of accuracies of the learning algorithms for this experiments, with uniform training instances. Accuracies of the same algorithms with skewed training data are shown in figure 7.13. These plots are self-explanatory, however, at this point we indicate few observations about these plots and this experiment. Following observations are used in later discussion in this chapter:

1. For uniform data, neural networks shows good accuracies for this experiment. However, its accuracies degrade in the case of skewed data.

2. Accuracies of N-Bins are not so good in the case of uniform data, but N-Bins out performs other algorithms by maintaining similar accuracies for skewed data.

3. For this experiment $P(success|refinedLimits) = 0.924$ (as mentioned above).

Since we are also interested in the response of the algorithms in presence of noise we also experiment with noisy data. For this purpose we generate the labels of the test instances by adding noise to the values of the parameters. That is, we first record the original values of the parameters in the test instances then we add noise to them and then we simulate. The labels of the test instances are then recorded based on the final states of the objects. This approach of adding noise to data is different from typical classification problem noise addition, which is achieved by randomly changing the labels in the training instances. We adopt this approach because of the settings of our problem. That is, in our problem we can always generate noise free data for training instances using the simulation, however in our settings the test instances (should ideally) come from sensory measurements of robots. Since these measurements include noise, therefore it is important to know the performances of the algorithms on the test data that includes noise. It is worth mentioning here, that different sources of training and test instances also imply that the distributions of the training and the test instances are different. Since the actual

distribution of the test instances is unknown to us, we choose uniform data for testing the accuracies of the learning algorithms.

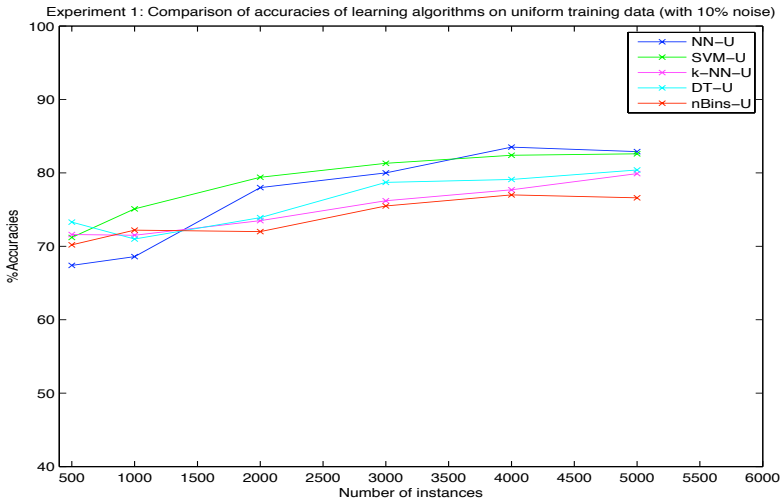


Figure 7.14: Comparison of accuracies of learning algorithms for experiment 1, with uniform training data and 10% noise in the test instances.

We repeat the testing of the algorithms with the noisy values of the parameters in the test instances, which are obtained by adding random values to the parameters. These values are up to 10% of the limits of the corresponding parameters. For such test instances, plots of the accuracies for uniform and skewed training data are shown in figures 7.14 and 7.15 respectively. It can be seen in these figures that there is no alarming change in the accuracies of the algorithms.

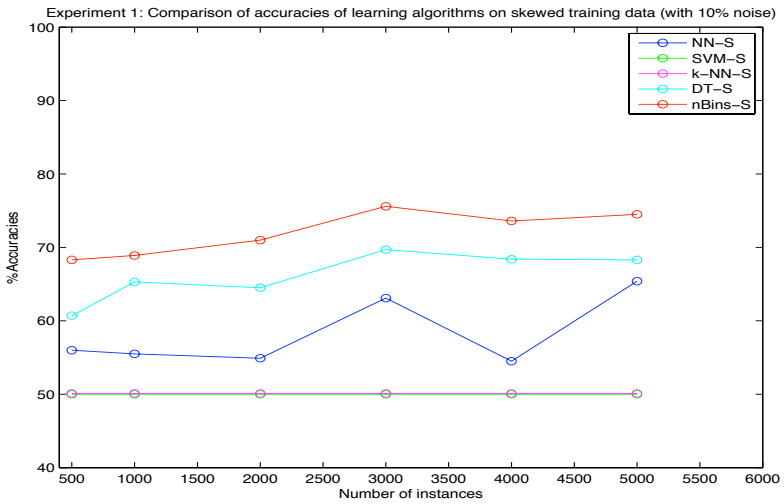


Figure 7.15: Comparison of accuracies of learning algorithms for experiment 1, with skewed training data and 10% noise in the test instances.

As mentioned in section 5.2.4, we do not give any absolute definition for the predicate `Allowed/3`. The results provided above correspond to *Condition-1, 2* and *3* mentioned in the description about *modifying the planning operator* in section 5.2.4.

7.2 Experiment 2

For this experiment and the experiments to follow, we keep the descriptive explanation of the results to minimum. We highlight the important issues only when we find it necessary. For the sake of conciseness we also skip few obvious results by referring to previously reported similar results. The results reported below follow the same organization pattern as those in experiment 1.

Specifications

In this experiment the simulation of the sample behavior shows that a `die` is dropped `over` the top surface of a `cube`, and in its final state the `die` stays `on` the `cube`. In the simulation, values of all the parameters in Δ are equal to 0.0 except the value of z, which is equal to 1.0. Thus, the simulation corresponds to the action of releasing the `die over` the center of the `cube`. Table 7.6 gives the specifications of the models of the objects used in the simulation.

Objects	**Dimensions** (m)	**Mass** (kg)
`Die`	0.14(h) $\times$ 0.14(l) $\times$ 0.14(w)	0.8
`Cube`	0.74(h) $\times$ 0.3(l) $\times$ 0.3(w)	∞

Table 7.6: Specifications of the objects for experiment 2.

Simulation description

Similar to the case of experiment 1, the approach places 588 markers on the `die` and finds $\mathtt{S}_{init}$ and $\mathtt{S}_{final}$ for the simulation. For this experiment, following are the expressions for the description of the behavior of the objects.

$\mathtt{S}_{init} \equiv$ `Over(Die, Cube)` $\wedge$ `Zz(Die, Cube)` $\wedge$ `None(Die)` $\wedge$ `Stationary-V(Die)` $\wedge$ `Stationary-AV(Die)`.

$\mathtt{S}_{final} \equiv$ `On(Die, Cube)` $\wedge$ `Zz(Die, Cube)` $\wedge$ `None(Die)` $\wedge$ `Stationary-V(Die)` $\wedge$ `Stationary-AV(Die)`.

The average simulation time (for a single simulation) for this experiment is 1.13 *sec*.

Limits of the parameters

Table 7.7 shows the initial and refined limits of the parameters of the simulation. These values are similar to those in table 7.2 in experiment 1. The only major difference is in the values of x and y parameters which get adjusted according to the dimensions of the top surface of the `cube`. Since the dimensions of the top surface of the `cube` are smaller (as compared to the dimensions of the `table` in experiment 1), it is more likely that when the `die` is dropped `over` the `cube`, it is not able to stay completely `on` the `cube`. This fact is

Parameter(s) δ(s)	**Lower limit** Initial	Refined	**Upper limit** Initial	Refined
x	-0.1875	-0.0654	0.1875	0.0762
y	-0.1875	-0.0622	0.1875	0.0680
z	0.8100	0.8283	1.0000	1.0000
ρ	-0.7853	-0.7849	0.7853	0.7850
θ	-0.7853	-0.7802	0.7853	0.7832
ϕ	-0.7853	-0.7841	0.7853	0.7844
$\dot{x}, \dot{y}, \dot{z}, \dot{\rho}, \dot{\theta}, \dot{\phi}$	-0.0010	-0.0009	0.0010	0.0009

Table 7.7: Initial and refined limits of the parameters in Δ, for experiment 2.

reflected in the values of table 7.8 which shows that the occurrences of negative examples are more frequent in this experiment than in experiment 1. Changes in the dimensions of the top surface of `object-2` also affect the extra evaluations of $\mathtt{S}_{init}$. Number of these evaluations is more than three times of the number of extra evaluations for experiment 1.

Stage of simulation process	+ve examples	−ve examples	Extra evaluations
Transition	324	177	9285
Termination	3345	1655	14217

Table 7.8: States of the simulation process for experiment 2.

From the numbers of negative and positive examples in the *termination* stage of the simulation process, we can see that:

$$P(success|refinedLimits) = \frac{3345}{5000} = 0.669$$

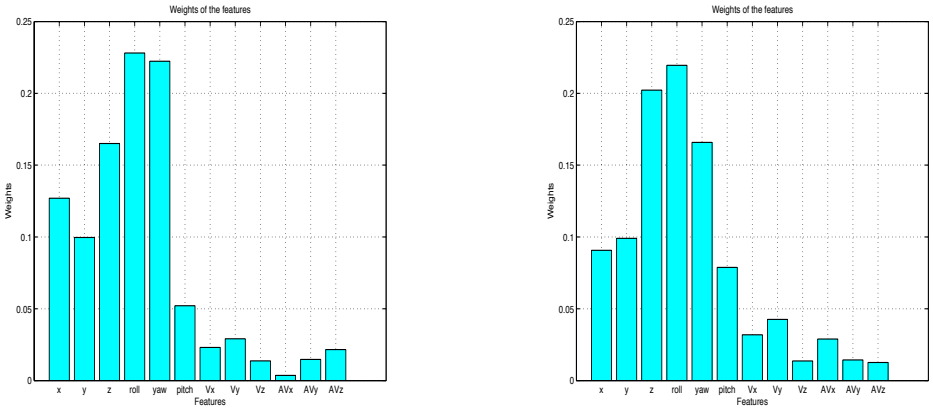
For this experiment when we generate 5000 examples of uniform data then the time taken by the simulation process is $TSim_u = 13.49\ hrs$ (i.e. 48554 *sec*). Whereas, the time taken for the generation of skewed data is $TSim_s = 8.93\ hrs$ (i.e. 32154 *sec*). From these values we have:

$$TSim_u = 1.62 \times TSim_s$$

Modification of planning operator

Figure 7.16 shows the weights of the parameters in this experiment according the uniform data with 5000 instances and skewed data with 3000 instance. From this figure we can again observe that the important parameters in figure (a) have high weights in figure (b). We do not show the distributions of the values of the parameters according to

instance labels for this experiment (and for the experiments to follow), as we did in figure 7.2 - 7.4 and 7.6 for experiment 1. This is because, the important aspect of the distributions (i.e. the patterns in the distribution) is also reflected in the weights of the parameters.



(a) 5000 instances (uniform data) (b) 3000 instances (skewed data)

Figure 7.16: Comparison of the weights of the parameters for two different sets of instances, for experiment 2.

Tables 7.9 and 7.10 show the information about the suggested values of the important parameters for the initial state of the objects in this experiment. The tables also show the limits of the bins from which the respective values are chosen (as the mean values of the bin limits) and the number of bins for each parameter. Just as in the case of experiment 1, it can be noticed here that for the cases of uniform and skewed data the suggested values of the corresponding parameters are not equal. However, if we choose (the complete set of) these values for the initial states of the objects in two separate simulations then both the simulations result in the desired behaviors of the objects. In other words, both of the (sets of) suggested values correspond to the same state of the objects, qualitatively.

Parameter(s) δ(s)	**Suggested values**	**Bin limits** Lower	Upper	**Number of bins**
x	-0.0108	-0.0187	-0.0029	9
y	-0.0083	-0.0168	0.0002	8
z	0.9753	0.9769	0.9835	11
ρ	0.1745	0.1454	0.2036	27
θ	-0.1052	-0.1575	-0.0529	15
ϕ	-0.2242	-0.3363	-0.1120	7

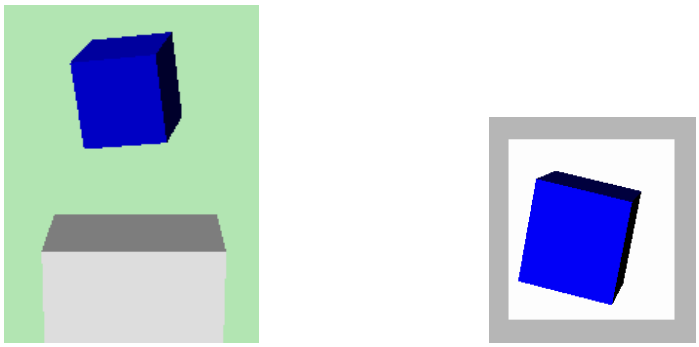
Table 7.9: Selected values and bins of the parameters based on uniform data (5000 instances) for experiment 2.

Parameter(s) δ(s)	**Suggested values**	**Bin limits** Lower	Upper	**Number of bins**
x	0.0085	0.0003	0.0167	8
y	0.0135	0.0066	0.0204	9
z	0.9827	0.9770	0.9884	15
ρ	0.1162	0.0871	0.1452	27
θ	0.1436	0.0725	0.2146	11
ϕ	-0.0001	-0.1121	0.1119	7

Table 7.10: Selected values and bins of the parameters based on skewed data (3000 instances) for experiment 2.

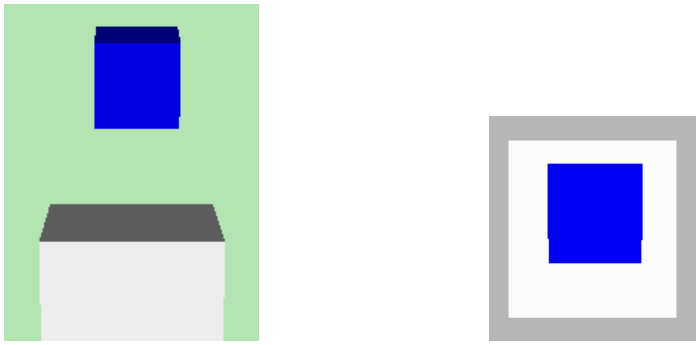
Figures 7.17 and 7.18 show the initial states of the objects when we use the suggested values shown in table 7.9 and 7.10 in the simulation process. Initial states of the objects with the worst values of the parameters, as estimated by N-Bins algorithm by using the uniform and the skewed data, are shown in figure 7.19 and 7.20. All these figures clearly show that our approach is not only able to calculate that what is the (approximate) best way to release the `die`, it is also able to explicate the worst way of releasing the `die` within the refined limits of the parameters. The similarity between the estimated worst initial states and the situation 1 shown in figure 4.1(a) in chapter 4 should be noticed here. We can see that qualitatively speaking all the three situations (i.e. the estimated worst initial states and the situation of figure 4.1(a)) are almost the same.

Figure 7.21 and 7.22 give the comparisons of accuracies of the learning algorithms with uniform and skewed training data respectively. As in the case of experiment 1, we would like the reader to especially notice the accuracies of neural networks (NN) and N-Bins in these figures. NN shows good performance for both the uniform and the skewed training data. On the other hand, accuracies of N-Bins are not very high in both the cases. However, N-Bins is able to maintain similar performance in both the figures. The reader is also reminded that for this experiment $P(success|refinedLimits) = 0.669$. In our experiments



(a) Front view (b) Top view

Figure 7.17: Suggested initial state for experiment 2, based on uniform data (5000 instances).

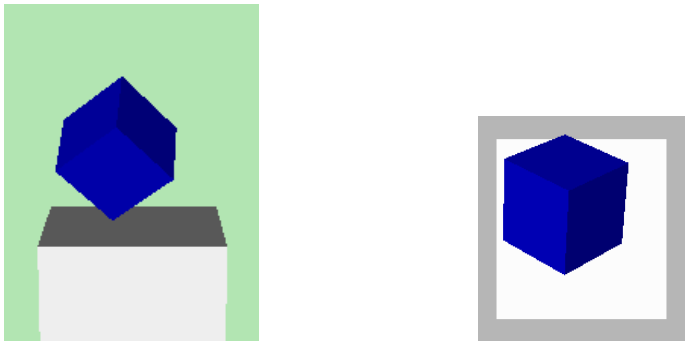


(a) Front view (b) Top view

Figure 7.18: Suggested initial state for experiment 2, based on skewed data (3000 instances).

we notice that generally for $0.3 < P(success|refinedLimits) < 0.7$, performance of NN is considerably good and consistent for both the cases of uniform and skewed data. NN also performs good for $P(success|refinedLimits) < 0.3$ or $P(success|refinedLimits) > 0.7$, if there is a strong dependence between the features (i.e. parameters) of the instances in the training data. Otherwise N-Bins out performs NN, especially in the cases of skewed data. These statements are further corroborated by the accuracy comparisons of other experiments presented in this chapter. However, unless necessary, we do not point out this fact explicitly in further descriptions of the plots of the accuracies of the learning algorithms.

For this experiment we also test the accuracies of the algorithms with the noisy test set. Comparisons of these accuracies are given in figure 7.23 and 7.24 for the uniform data and the skewed data. These figures show that (similar to the case of experiment 1) there is no dramatic change in the results. This fact is true in general for all the experiments reported in the thesis. Therefore in the experiments to follow, we do not give separate plots of the accuracies of the algorithms for noise free and noisy test instances. Instead we report only the accuracies with noisy test set. However, we choose the figure of noise to be 5% of the limits of the corresponding parameters.



(a) Front view (b) Top view

Figure 7.19: Estimated worst initial state for experiment 2, using uniform data (5000 instances).

(a) Front view (b) Top view

Figure 7.20: Estimated worst initial state for experiment 2, using skewed data (3000 instances).

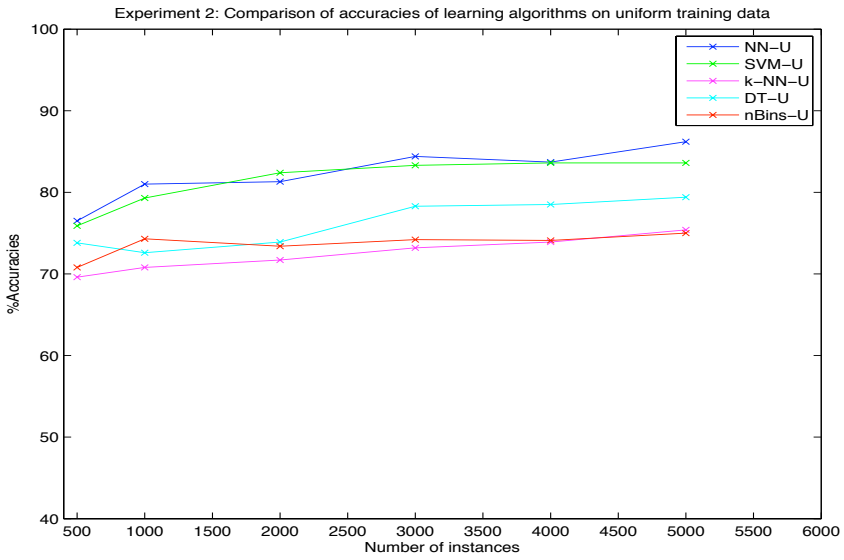


Figure 7.21: Comparison of accuracies of learning algorithms for experiment 2, with uniform training data.

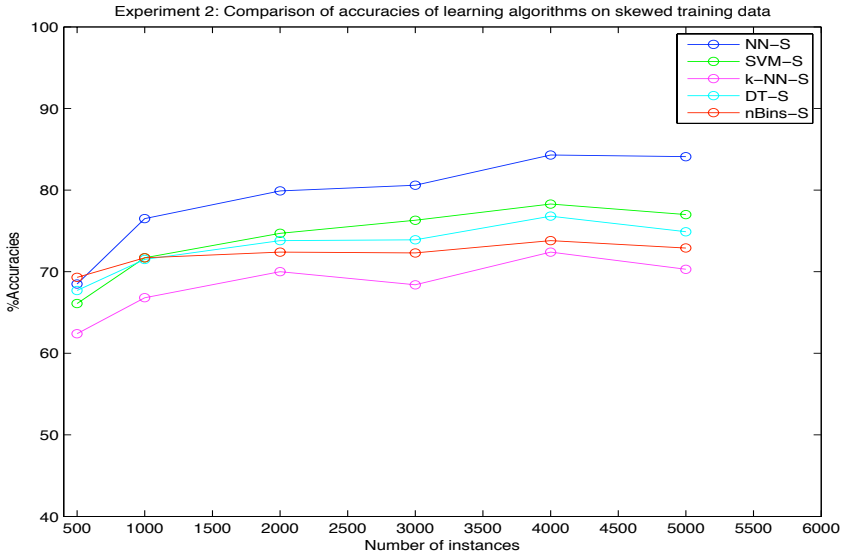


Figure 7.22: Comparison of accuracies of learning algorithms for experiment 2, with skewed training data.

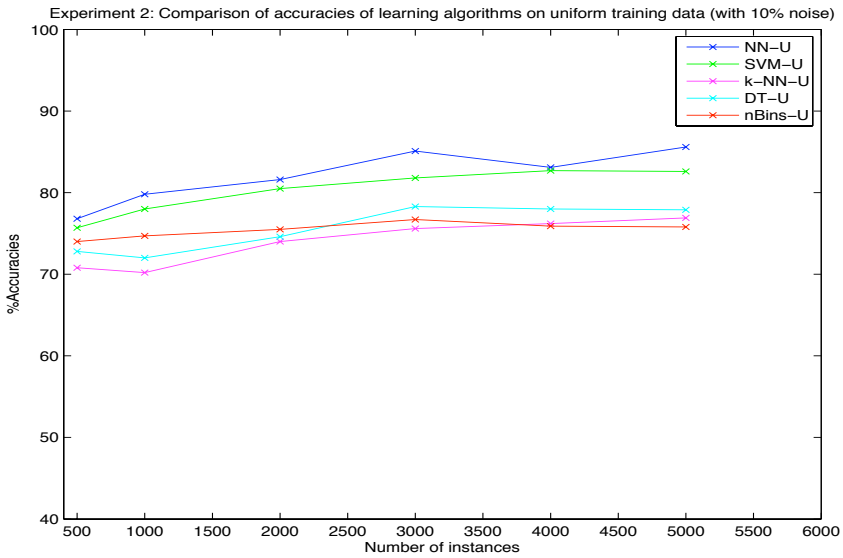


Figure 7.23: Comparison of accuracies of learning algorithms for experiment 2, with uniform training data and 10% noise in the test instances.

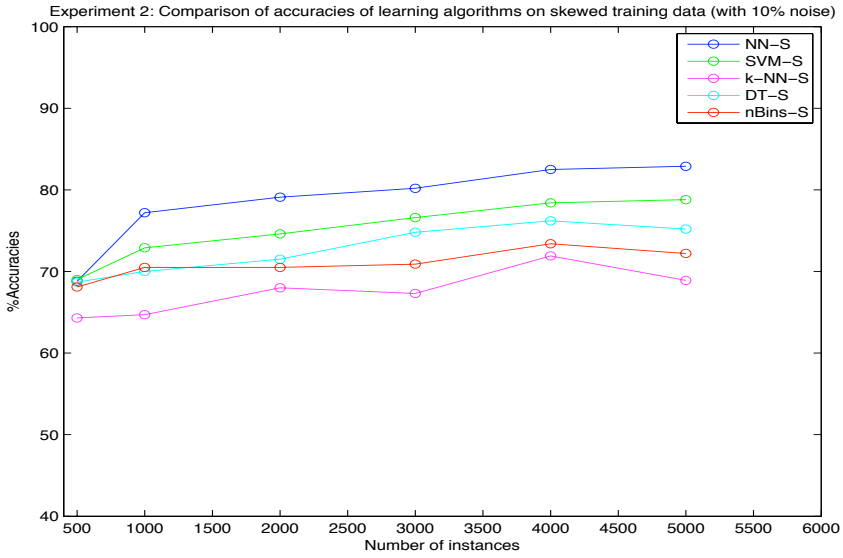


Figure 7.24: Comparison of accuracies of learning algorithms for experiment 2, with skewed training data and 10% noise in the test instances.

7.3 Experiment 3

From our previous work (Akhtar [2011]) and observation (5) in chapter 4 we know that occurrence of unknown external faults is dependent on the intrinsic properties (e.g. shape and size) of the objects. Therefore, we find it is necessary to analyze the proposed approach with the help of different objects. In this experiment and experiment 4 we analyze the effects of changing the geometric shape[5] of the top surface of `object-2` on our resutls.

Specifications

For this experiment the simulation of the sample behavior shows that a `die` is dropped `over` the center of a `round table`. In the final state of the simulation the `die` stays `on` the `round table`. Values of the parameters for the simulation are same as those for the first two experiments. Specifications of the objects are shown in table 7.11.

Objects	**Dimensions** (m)	**Mass** (kg)
`Die`	0.14(h) × 0.14(l) × 0.14(w)	0.8
`Round Table`	0.74(h), 0.35(r)	∞

Table 7.11: Specifications of the objects for experiment 3.

Simulation description

The approach finds the two FOL expressions given below to describe the behavior of the objects. This is done by placing 588 markers on the `die` and simulating the objects. The average time for a single simulation for this experiment is 1.0 *sec*.

$S_{init} \equiv$ `Over(Die, RoundTable)` $\wedge$ `Zz(Die, RoundTable)` $\wedge$ `None(Die)`
$\wedge$ `Stationary-V(Die)` $\wedge$ `Stationary-AV(Die)`.

$S_{final} \equiv$ `On(Die, RoundTable)` $\wedge$ `Zz(Die, RoundTable)` $\wedge$ `None(Die)`
$\wedge$ `Stationary-V(Die)` $\wedge$ `Stationary-AV(Die)`.

Limits of the parameters

For this experiment, values of the initial and the refined limits of the parameters are given in table 7.12. From these values it is clear that at this stage of the approach, treatment of the circular top of the `table` is no different than a square or a rectangular top of the `table`. That is, the refined values of the limits of x and y are not taking any special care of the dependence[6] between these two parameters. One might expect that

[5] In the first two experiments the differences in the shapes of `object-2` is only that of dimensions. Here the geometry itself is changed.

[6] We know, for a circle with center at origin $\sqrt{x^2 + y^2} = r$. Ideally the approach should refine the limits of x and y by using an inequality like $\sqrt{x^2 + y^2} \leq r$. However, the approach does not make use of any explicit equation or inequality.

this can results in generation of a large number of negative examples for this experiment. However, table 7.13 suggests otherwise. From this table it is clear that in this experiment frequency of negative example generation in the simulation process is not much different than those of earlier experiments. The simple reason behind this fact is that the initial states of the objects with extreme x, y values (e.g. -0.251,-0.249) result in $\mathtt{S}_{init}$ with `P1` $\equiv$ `Partially-Over/2` instead of `Over/2`. This makes the simulation process to drop these values of the parameters and select new values for which `P1` $\equiv$ `Over/2`. Thus, the simulation process is able to select the combinations of x, y values from the refined limits that respect the dependence between the parameters, and this does not require any explicit equation of the top surface of `object-2`.

Parameter(s) δ(s)	**Lower limit** Initial	Refined	**Upper limit** Initial	Refined
x	-0.4375	-0.2525	0.4375	0.2469
y	-0.4375	-0.2495	0.4375	0.2475
z	0.8100	0.8211	1.0000	1.0000
ρ	-0.7853	-0.7842	0.7853	0.7847
θ	-0.7853	-0.7813	0.7853	0.7821
ϕ	-0.7853	-0.7840	0.7853	0.7814
$\dot{x}, \dot{y}, \dot{z}, \dot{\rho}, \dot{\theta}, \dot{\phi}$	-0.0010	-0.0009	0.0010	0.0009

Table 7.12: Initial and refined limits of the parameters in Δ, for experiment 3.

Stage of simulation process	+ve examples	−ve examples	Extra evaluations
Transition	645	51	2975
Termination	4632	368	5629

Table 7.13: States of the simulation process for experiment 3.

From table 7.13, we can see that:

$$P(success|refinedLimits) = \frac{4632}{5000} = 0.926$$

Furthermore, for this experiment $TSim_u = 34.3\ hrs$ (i.e. 123608 *sec*) and $TSim_s = 5.88$ hrs (i.e. 21198 *sec*). This makes:

$$TSim_u = 5.83 \times TSim_s$$

Modification of planning operator

Tables 7.14 and 7.15 show the values of the parameters suggested by the N-Bins algorithm for the initial state of the objects in this experiment. The tables also show the limits of the bins and the total number of bins for each parameter. Weights of the parameters based on the uniform and the skewed data are shown in figure 7.25.

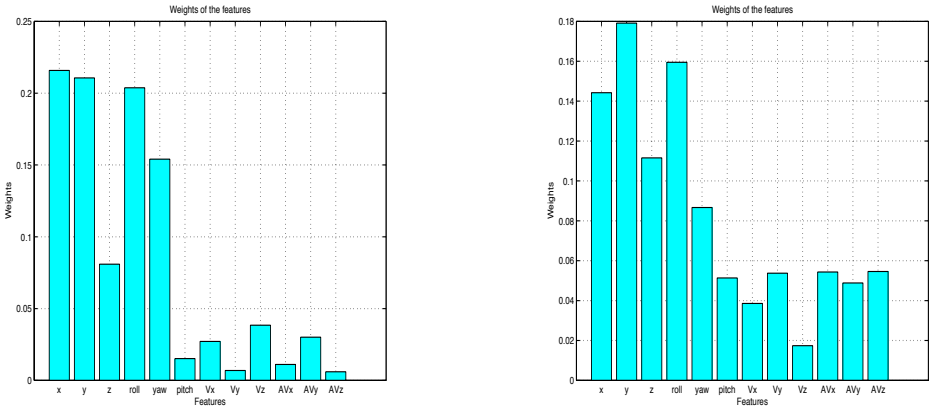
Parameter(s) δ(s)	**Suggested values**	**Bin limits** Lower	Upper	**Number of bins**
x	-0.0398	-0.0490	-0.0305	27
y	-0.0322	0.0156	0.0487	15
z	0.9884	0.9769	1.0000	8
ρ	-0.1428	-0.2142	-0.0714	11
θ	0.0001	-0.0872	0.0873	9
ϕ	-0.3926	-0.5235	-0.2617	6

Table 7.14: Selected values and bins of the parameters based on uniform data (5000 instances) for experiment 3.

Parameter(s) δ(s)	**Suggested values**	**Bin limits** Lower	Upper	**Number of bins**
x	0.0378	0.0151	0.0604	11
y	0.0754	0.0658	0.0849	27
z	0.9693	0.9592	0.9795	9
ρ	-0.0012	-0.0534	0.0511	15
θ	0.0977	-0.0003	0.1957	8
ϕ	0.6487	0.5184	0.7790	6

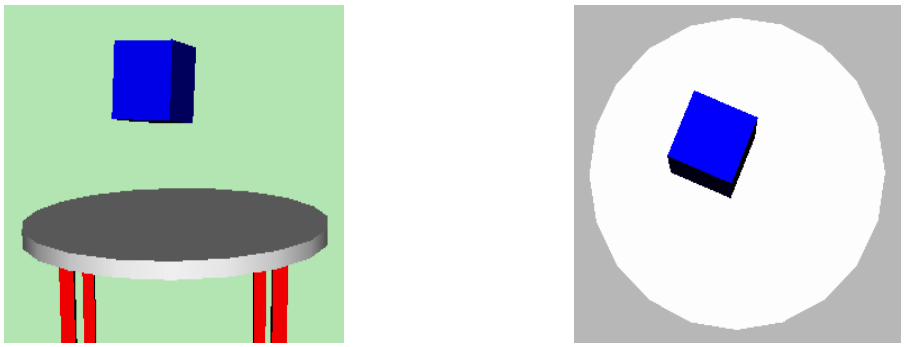
Table 7.15: Selected values and bins of the parameters based on skewed data (3000 instances) for experiment 3.

Figure 7.26 to 7.29 show the suggested initial states and estimated worst initial states for this experiment based on uniform data (5000 instances) and skewed data (3000 instances). For this experiment, accuracies of the learning algorithms for both the data sets are shown in figure 7.30 and 7.31.



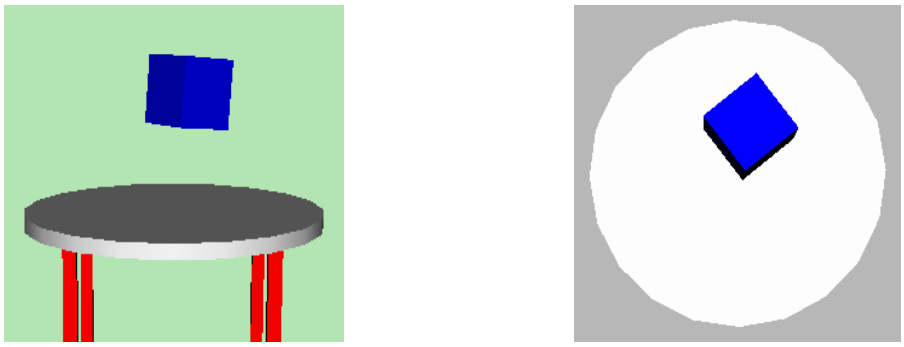
(a) 5000 instances (uniform data) (b) 3000 instances (skewed data)

Figure 7.25: Comparison of the weights of the parameters for two different sets of instances, for experiment 3.



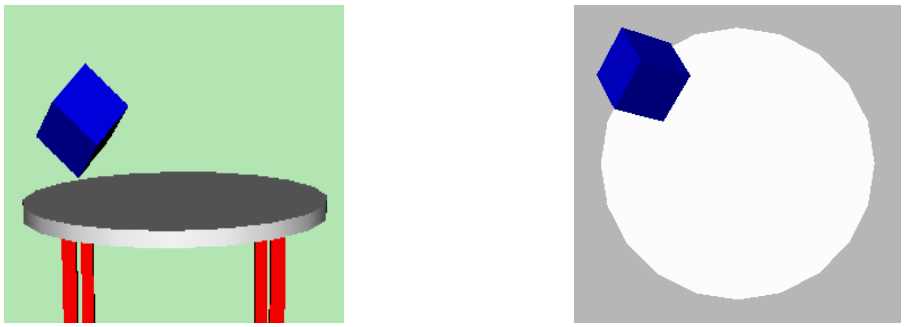
(a) Front view (b) Top view

Figure 7.26: Suggested initial state for experiment 3, based on uniform data (5000 instances).



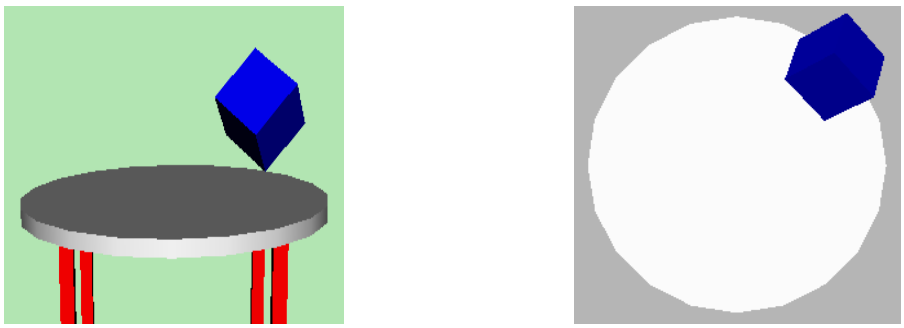
(a) Front view (b) Top view

Figure 7.27: Suggested initial state for experiment 3, based on skewed data (3000 instances).



(a) Front view (b) Top view

Figure 7.28: Estimated worst initial state for experiment 3, using uniform data (5000 instances).



(a) Front view (b) Top view

Figure 7.29: Estimated worst initial state for experiment 3, using skewed data (3000 instances).

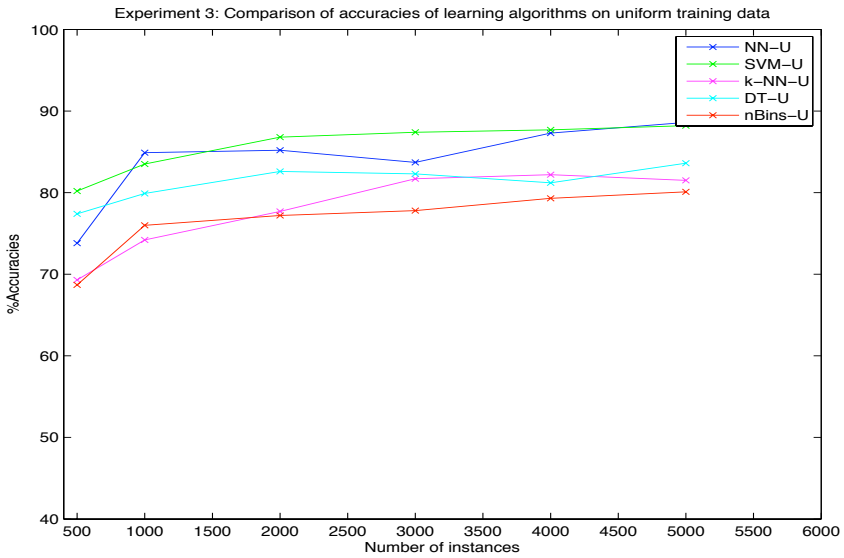


Figure 7.30: Comparison of accuracies of learning algorithms for experiment 3, with uniform training data.

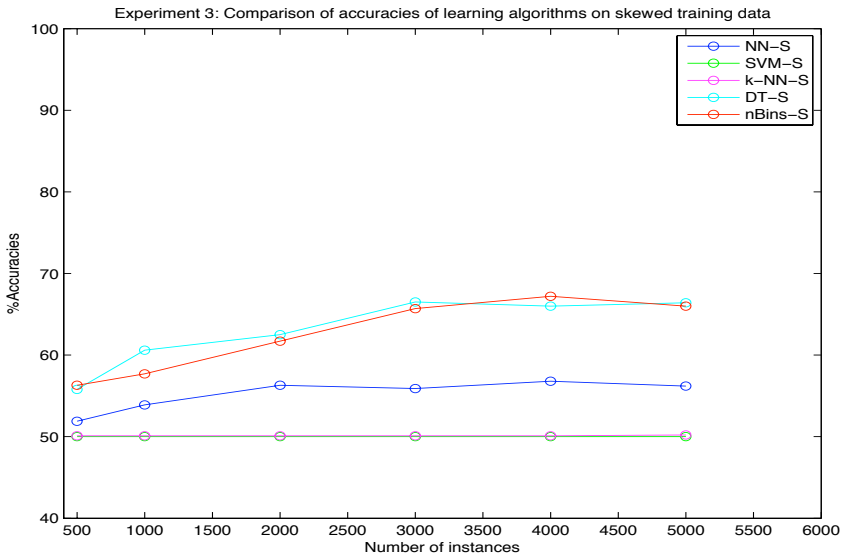


Figure 7.31: Comparison of accuracies of learning algorithms for experiment 3, with skewed training data.

7.4 Experiment 4

Specifications

In this experiment, the simulation of the sample behavior shows that a `die` is dropped `over` a `table` *near the edge*. The table's top surface is composed of semi-circles and a rectangle (see figure 7.33). Dimensions of the `table` are given below.

Objects	**Dimensions** (m)	**Mass** (kg)
`Die`	0.14(h) × 0.14(l) × 0.14(w)	0.8
`Table`	0.74(h) × 0.6(l) × 1.2(w)	∞

Table 7.16: Specifications of the objects for experiment 4.

For this experiment $x = -0.2, y = -0.4$ and $z = 1$ and all other parameters in Δ are equal to 0.0. This means that the `die` is released in a straight orientation near the left circular edge of the table. In the final state, the die remains `on` the `table`.

Simulation description

The FOL sentences given below show the description of the simulation found by the approach.

$S_{init} \equiv$ `Over(Die, Table)` $\wedge$ `Nn(Die, Table)` $\wedge$ `None(Die)` $\wedge$ `Stationary-V(Die)` $\wedge$ `Stationary-AV(Die)`.

$S_{final} \equiv$ `On(Die, Table)` $\wedge$ `Nn(Die, Table)` $\wedge$ `None(Die)` $\wedge$ `Stationary-V(Die)` $\wedge$ `Stationary-AV(Die)`. For this experiment the average time for a single simulation is 1.21 *sec*.

Limits of the parameters

Tables 7.17 and 7.18 show the values of the limits of the parameters and the details of the distribution of the examples for the simulation process. Based on the information in table 7.18 we find that:

$$P(success|refinedLimits) = \frac{4651}{5000} = 0.930$$

For this experiment, $TSim_u = 39.56\ hrs$ (i.e. 142425 *sec*) and $TSim_s = 7.67\ hrs$ (i.e. 27642 *sec*). Which makes:

$$TSim_u = 5.16 \times TSim_s$$

Parameter(s) δ(s)	**Lower limit** Initial	Refined	**Upper limit** Initial	Refined
x	-0.7500	-0.4893	-0.0001	-0.0002
y	-0.3750	-0.2100	-0.0001	-0.0002
z	0.8100	0.8304	1.0000	0.9998
ρ	-0.7853	-0.7841	0.7853	0.7842
θ	-0.7853	-0.7849	0.7853	0.7845
ϕ	-0.7853	-0.7847	0.7853	0.7838
$\dot{x}, \dot{y}, \dot{z}, \dot{\rho}, \dot{\theta}, \dot{\phi}$	-0.0010	-0.0009	0.0010	0.0009

Table 7.17: Initial and refined limits of the parameters in Δ, for experiment 4.

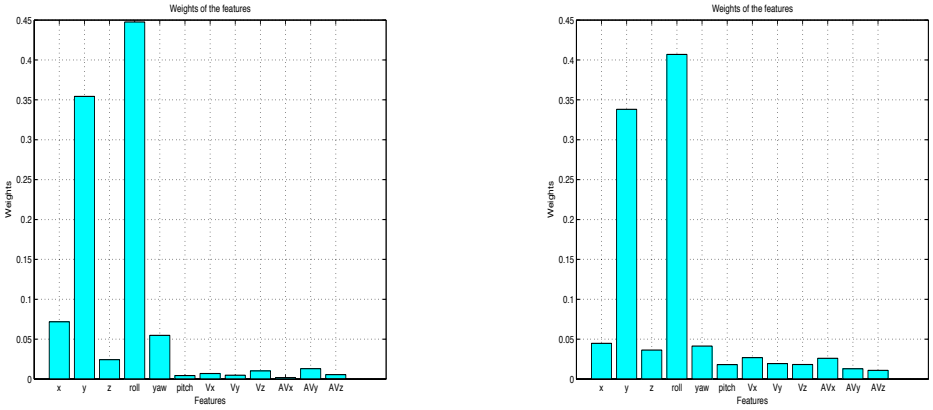
Stage of simulation process	+ve examples	−ve examples	Extra evaluations
Transition	625	51	1990
Termination	4651	349	3447

Table 7.18: States of the simulation process for experiment 4.

Modification of planning operator

Figure 7.32 shows the comparison of the weights of the parameters for the uniform data and the skewed data generated by the simulation process in this experiment. Information regarding the suggested values of the parameters for initial state and bins of the values, is given in tables 7.19 and 7.20 for both the data sets. Figures 7.33 to 7.36 show the initial states of the objects, using the suggested values and estimated worst values of the parameters. It is noticeable in figures 7.33 and 7.34 that despite the fact that the simulation of the sample behavior shows that the `die` is dropped near the corner of the `table`, the suggested initial positions for the `die` are near the center of the `table`, for both the data sets. Furthermore, N-Bins algorithms suggests that (for both the data sets) the worst way to perform the action is to drop the `die` on its edge near the *circular* boundary of the `table`. This is an example of the potential ability of N-Bins algorithm to extract the understanding of the behaviors of the objects from labelled examples.

Comparisons of accuracies of the learning algorithms for this experiment are given in figure 7.37 and 7.38. Figure 7.37 gives the comparison using uniform data set for training, whereas figure 7.38 gives the comparison based on skewed training data.



(a) 5000 instances (uniform data) (b) 3000 instances (skewed data)

Figure 7.32: Comparison of the weights of the parameters for two different sets of instances, for experiment 4.

Parameter(s) δ(s)	**Suggested values**	**Bin limits** Lower	Upper	**Number of bins**
x	-0.0670	-0.0892	-0.0447	11
y	-0.0072	-0.0142	-0.0002	15
z	0.9469	0.9364	0.9575	8
ρ	0.0001	-0.0290	0.0291	27
θ	-0.0002	-0.0874	0.0870	9
ϕ	-0.3926	-0.5233	-0.2619	6

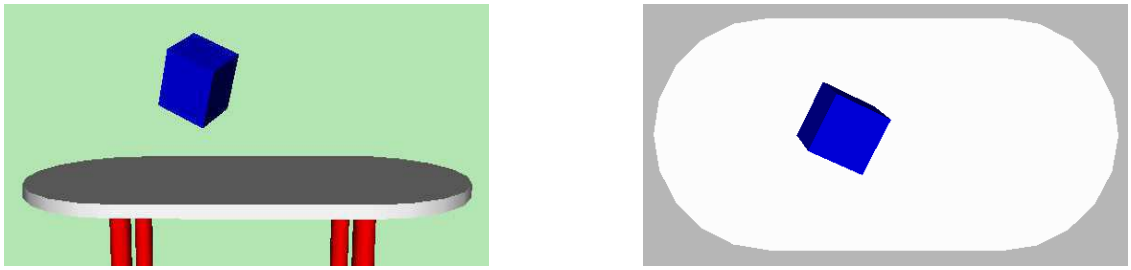
Table 7.19: Selected values and bins of the parameters based on uniform data (5000 instances) for experiment 4.

Parameter(s) δ(s)	**Suggested values**	**Bin limits** Lower	Upper	**Number of bins**
x	-0.1131	-0.1357	-0.0905	11
y	-0.0074	-0.0145	-0.0003	15
z	0.9669	0.9560	0.9778	8
ρ	0.2323	0.2033	0.2614	27
θ	0.3500	0.2631	0.4369	9
ϕ	-0.3918	-0.5224	-0.2612	6

Table 7.20: Selected values and bins of the parameters based on skewed data (3000 instances) for experiment 4.

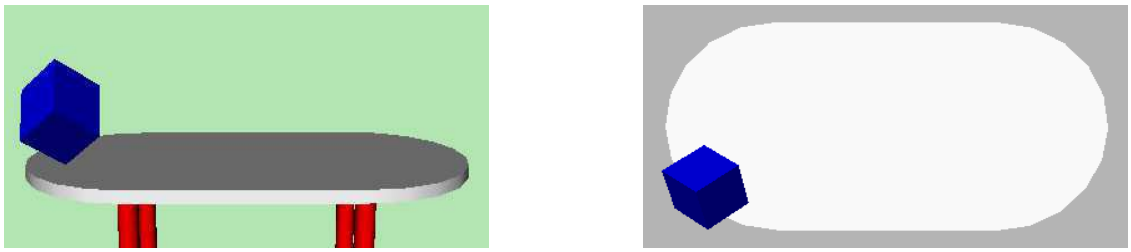
(a) Front view (b) Top view

Figure 7.33: Suggested initial state for experiment 4, based on uniform data (5000 instances).



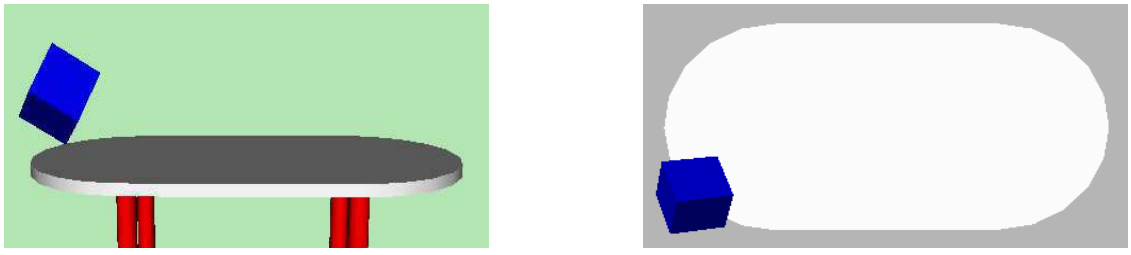
(a) Front view (b) Top view

Figure 7.34: Suggested initial state for experiment 4, based on skewed data (3000 instances).



(a) Front view (b) Top view

Figure 7.35: Estimated worst initial state for experiment 4, using uniform data (5000 instances).



(a) Front view (b) Top view

Figure 7.36: Estimated worst initial state for experiment 4, using skewed data (3000 instances).

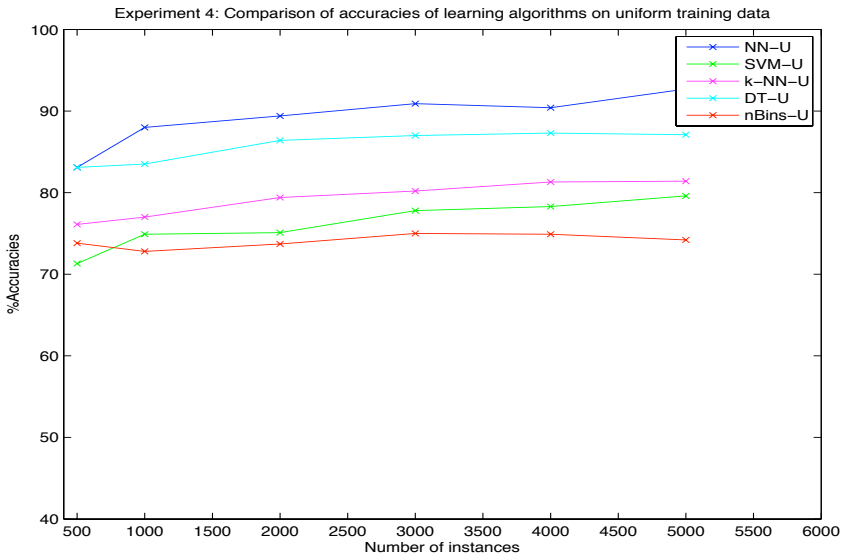


Figure 7.37: Comparison of accuracies of learning algorithms for experiment 4, with uniform training data.

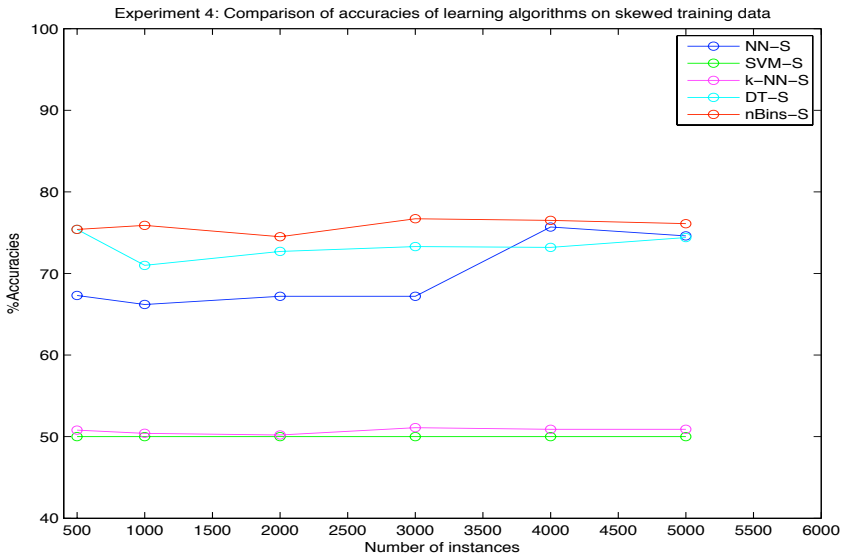


Figure 7.38: Comparison of accuracies of learning algorithms for experiment 4, with skewed training data.

7.5 Experiment 5

This experiment along with experiment 6 shows the effects of changing `object-1` on the results of our approach.

Specifications

For this experiment simulation of the sample behavior shows that a `pack`[7] (i.e. a carton) is dropped `over` the center of a `table`. Specification of the objects used in the simulation are shown in table 7.21.

Objects	**Dimensions** (m)	**Mass** (kg)
`Pack`	0.195(h) × 0.09(l) × 0.06(w)	1.0
`Table`	0.74(h) × 0.6(l) × 0.6(w)	∞

Table 7.21: Specifications of the objects for experiment 5.

Simulation description

In order to get the description of the simulation 338 markers are placed on the surface of the `pack`. The FOL expressions below show the description of the simulation obtained for this experiment.

S_{init} ≡ `Over(Pack, Table)` ∧ `Zz(Pack, Table)` ∧ `StraightAlong-z(Pack)` ∧ `Stationary-V(Pack)` ∧ `Stationary-AV(Pack)`.

S_{final} ≡ `On(Pack, Table)` ∧ `Zz(Pack, Table)` ∧ `StraightAlong-z(Pack)` ∧ `Stationary-V(Pack)` ∧ `Stationary-AV(Pack)`.

On average, a single simulation took 1.01 *sec* for this experiment.

Limits of the parameters

Table 7.22 shows the initial and refined limits of the parameters for this experiment. Information regarding the distributions of examples is shown in table 7.23. According to this table:

$$P(success|refinedLimits) = \frac{4515}{5000} = 0.903$$

Furthermore, for this experiment $TSim_u = 13.59\ hrs$ (i.e. 48940 *sec*) and $TSim_s = 2.89\ hrs$ (i.e. 10421 *sec*), which makes:

$$TSim_u = 4.7 \times TSim_s$$

[7] Modeled as a solid object.

Parameter(s) δ(s)	Lower limit Initial	Lower limit Refined	Upper limit Initial	Upper limit Refined
x	-0.3750	-0.2587	0.3750	0.2582
y	-0.3750	-0.2450	0.3750	0.2435
z	0.8299	0.8306	1.020	1.0198
ρ	-0.1963	-0.1959	0.1963	0.1957
θ	-0.1963	-0.1951	0.1963	0.1941
ϕ	-0.7853	-0.7785	0.7853	0.7847
$\dot{x}, \dot{y}, \dot{z}, \dot{\rho}, \dot{\theta}, \dot{\phi}$	-0.0010	-0.0009	0.0010	0.0009

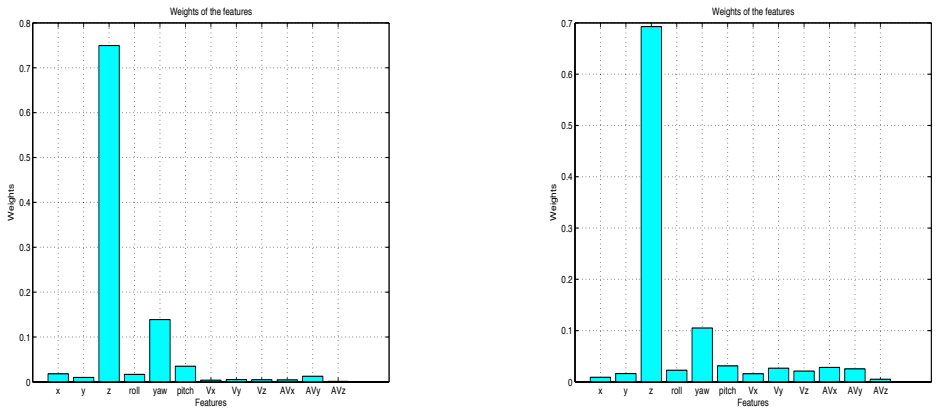
Table 7.22: Initial and refined limits of the parameters in Δ, for experiment 5.

Stage of simulation process	+ve examples	−ve examples	Extra evaluations
Transition	428	73	622
Termination	4515	485	859

Table 7.23: States of the simulation process for experiment 5.

Modification of planning operator

Comparison of the weights of the parameters for the uniform and skewed data for this experiment is given in figure 7.39, whereas tables 7.24 and 7.25 give the details about the suggested values of the parameters and the bins, for both the data sets. Initial states of the objects according to the suggested values of the parameters and their estimated worst values are shown in figure 7.40 - 7.43. Plots of the accuracies of the learning algorithms based on both the training data sets are shown in figure 7.44 and 7.45.



(a) 5000 instances (uniform data) (b) 3000 instances (skewed data)

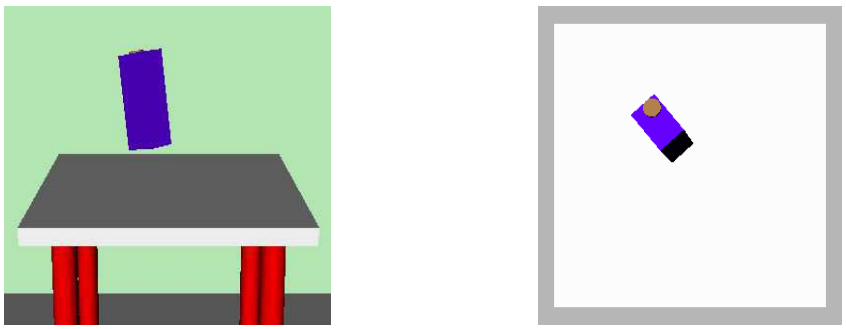
Figure 7.39: Comparison of the weights of the parameters for two different sets of instances, for experiment 5.

Parameter(s) δ(s)	**Suggested values**	**Bin limits** Lower	Upper	**Number of bins**
x	-0.0577	-0.0864	-0.0290	9
y	0.0691	0.0342	0.1040	7
z	0.8902	0.8867	0.8937	27
ρ	-0.1715	-0.1960	-0.1470	8
θ	-0.0265	-0.0395	-0.0135	15
ϕ	0.7137	0.6426	0.7848	11

Table 7.24: Selected values and bins of the parameters based on uniform data (5000 instances) for experiment 5.

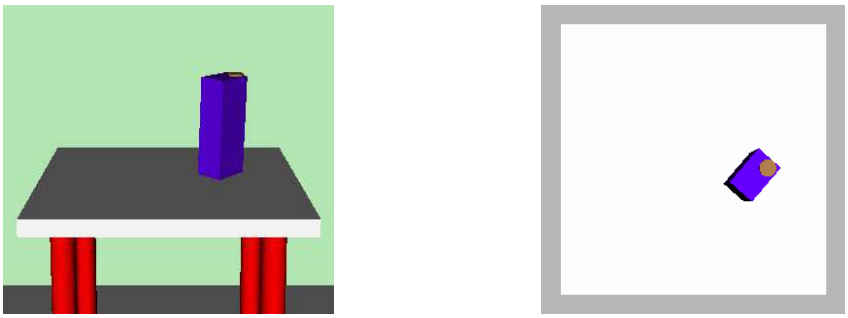
Parameter(s) δ(s)	**Suggested values**	**Bin limits** Lower	Upper	**Number of bins**
x	0.1235	0.0804	0.1667	6
y	-0.0438	-0.0853	-0.0022	6
z	0.8478	0.8443	0.8513	27
ρ	-0.0002	-0.0281	0.0278	7
θ	-0.0517	-0.0648	-0.0387	15
ϕ	-0.7136	-0.7849	-0.6423	11

Table 7.25: Selected values and bins of the parameters based on skewed data (3000 instances) for experiment 5.



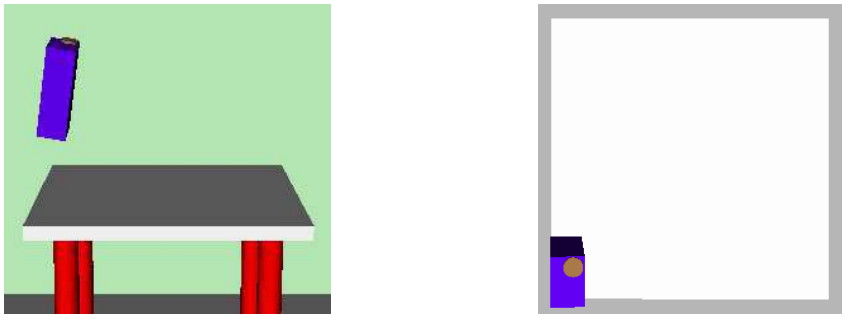
(a) Front view (b) Top view

Figure 7.40: Suggested initial state for experiment 5, based on uniform data (5000 instances).



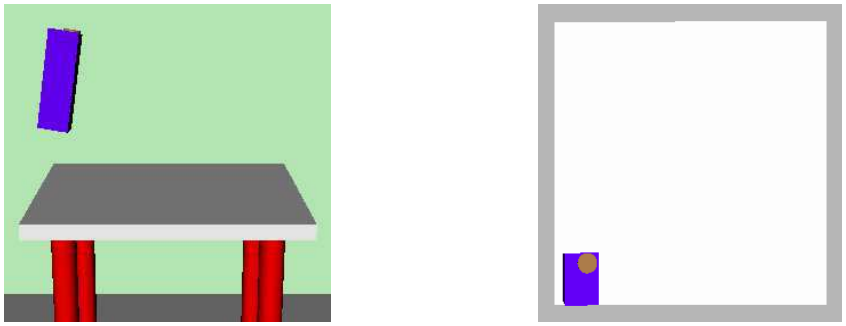
(a) Front view (b) Top view

Figure 7.41: Suggested initial state for experiment 5, based on uniform data (5000 instances).



(a) Front view (b) Top view

Figure 7.42: Estimated worst initial state for experiment 5, using uniform data (5000 instances).



(a) Front view (b) Top view

Figure 7.43: Estimated worst initial state for experiment 5, using uniform data (5000 instances).

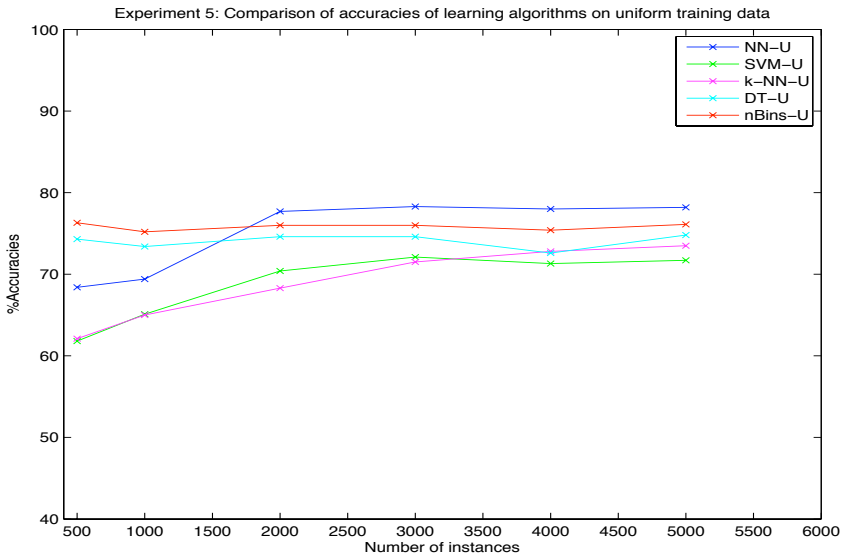


Figure 7.44: Comparison of accuracies of learning algorithms for experiment 5, with uniform training data.

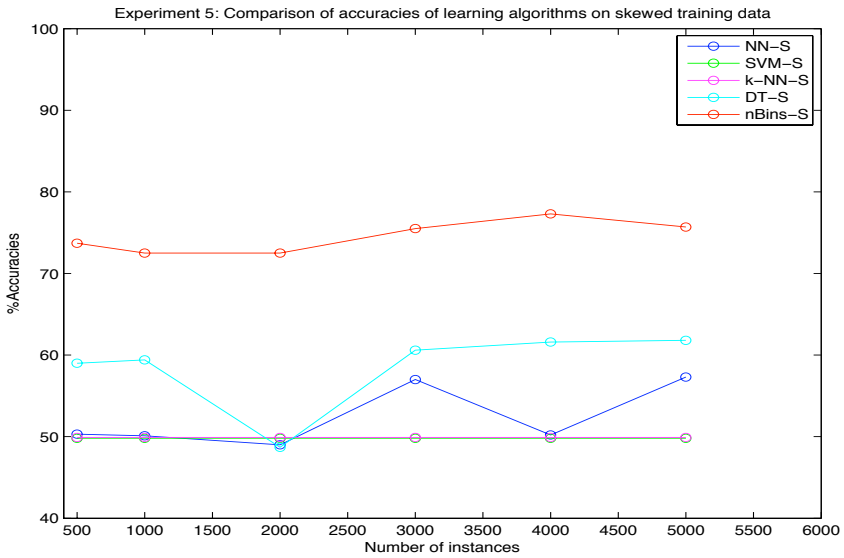


Figure 7.45: Comparison of accuracies of learning algorithms for experiment 5, with skewed training data.

7.6 Experiment 6

Specifications

The simulation of the sample behavior for this experiment shows that a `bottle` is dropped `over` the center of a `cube` and the `bottle` stands straight `on` the `cube` in the final state of the simulation. Specifications of the objects for this experiment are shown in the table below.

Objects	**Dimensions** (m)	**Mass** (kg)
`Bottle`	0.185(h), 0.05(r) × 0.06(w)	0.5
`Cube`	0.74(h) × 0.3(l) × 0.3(w)	∞

Table 7.26: Specifications of the objects for experiment 6.

Simulation description

Following is the description of the simulation found by our approach for this experiment.

S_{init} ≡ `Over(Bottle, Cube)` ∧ `Zz(Bottle, Cube)` ∧ `StraightAlong-z(Bottle)` ∧ `Stationary-V(Bottle)` ∧ `Stationary-AV(Bottle)`.

S_{final} ≡ `On(Bottle, Cube)` ∧ `Zz(Bottle, Cube)` ∧ `StraightAlong-z(Bottle)` ∧ `Stationary-V(Bottle)` ∧ `Stationary-AV(Bottle)`.

To find the above mentioned sentences, 178 markers were placed on the `bottle` and a single simulation took 1.14 *sec* on average.

Limits of the parameters

Table 7.27 shows the information on the limits of the parameters found by the approach for this experiment. The information regarding the distributions of the examples generated in the simulation process is given in table 7.28. From this table:

$$P(success|refinedLimits) = \frac{4417}{5000} = 0.883$$

For this experiment $TSim_u = 7.72\ hrs$ (i.e. 27795 *sec*) and $TSim_s = 2.9\ hrs$ (i.e. 10439 *sec*), which makes:

$$TSim_u = 2.66 \times TSim_s$$

Parameter(s) δ(s)	Lower limit Initial	Lower limit Refined	Upper limit Initial	Upper limit Refined
x	-0.1875	-0.1221	0.1875	0.1244
y	-0.1875	-0.1224	0.1875	0.1196
z	0.8100	0.8102	1.000	0.9999
ρ	-0.1963	-0.1953	0.1963	0.1956
θ	-0.1963	-0.1952	0.1963	0.1954
ϕ	-0.7853	-0.7851	0.7853	0.7763
$\dot{x}, \dot{y}, \dot{z}, \dot{\rho}, \dot{\theta}, \dot{\phi}$	-0.0010	-0.0009	0.0010	0.0009

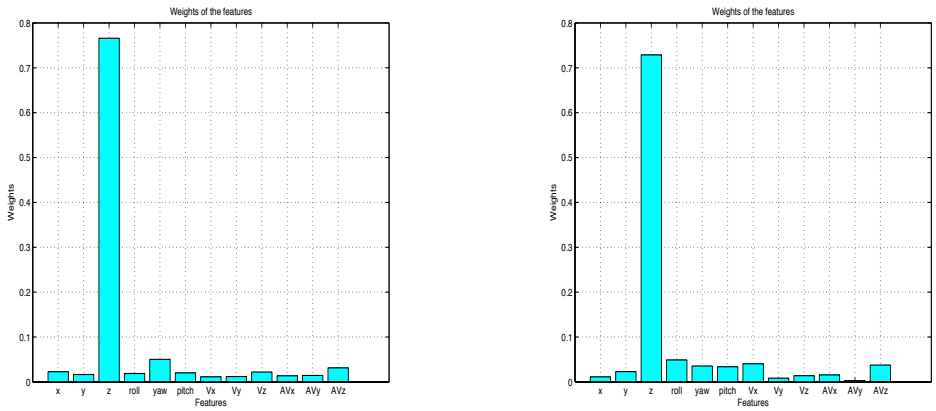
Table 7.27: Initial and refined limits of the parameters in Δ, for experiment 6.

Stage of simulation process	+ve examples	−ve examples	Extra evaluations
Transition	414	87	719
Termination	4417	583	1172

Table 7.28: States of the simulation process for experiment 6.

Modification of planning operator

Figure 7.46 shows the comparison of the weights of the parameters for the uniform and the skewed data sets for this experiment. Information regarding the suggested values of the parameters and the bins used to find these values is given in table 7.29 and 7.30. Table 7.29 shows the values for uniform data and table 7.30 shows the values for skewed data. For this experiment initial states of the object with suggested values and estimated worst values of the parameters for both the data sets are given in figure 7.47 - 7.50. Furthermore, figure 7.51 and 7.52 give the comparisons of the accuracies of the learning algorithms for uniform and skewed training data respectively.



(a) 5000 instances (uniform data) (b) 3000 instances (skewed data)

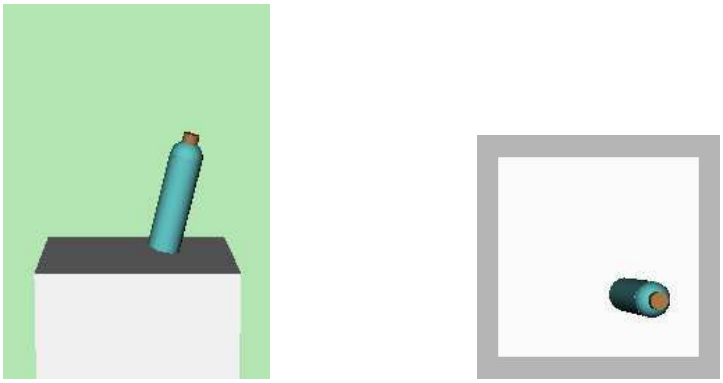
Figure 7.46: Comparison of the weights of the parameters for two different sets of instances, for experiment 6.

Parameter(s) δ(s)	**Suggested values**	**Bin limits** Lower	Upper	**Number of bins**
x	0.0560	0.0423	0.0697	9
y	-0.0691	-0.0821	-0.0418	6
z	0.8278	0.8243	0.8313	27
ρ	0.1677	0.1398	0.1956	7
θ	0.1824	0.1694	0.1954	15
ϕ	0.4417	0.3302	0.5533	7

Table 7.29: Selected values and bins of the parameters based on uniform data (5000 instances) for experiment 6.

Parameter(s) δ(s)	**Suggested values**	**Bin limits** Lower	Upper	**Number of bins**
x	-0.0214	-0.0417	-0.0011	6
y	0.0362	0.0187	0.0538	7
z	0.8418	0.8383	0.8453	27
ρ	0.1566	0.1436	0.1696	15
θ	-0.1717	-0.1962	-0.1472	8
ϕ	0.6733	0.5613	0.7853	7

Table 7.30: Selected values and bins of the parameters based on skewed data (3000 instances) for experiment 6.

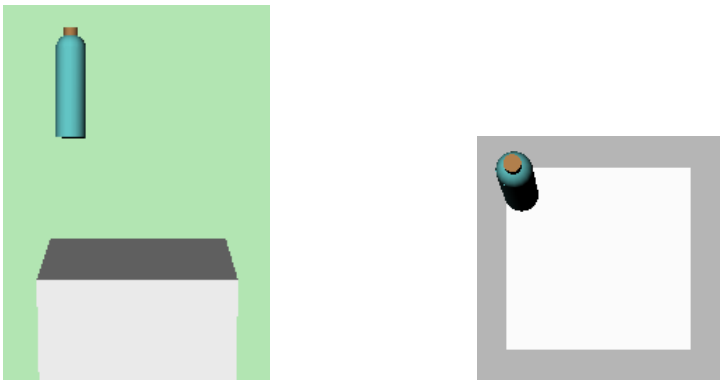


(a) Front view (b) Top view

Figure 7.47: Suggested initial state for experiment 6, based on uniform data (5000 instances).

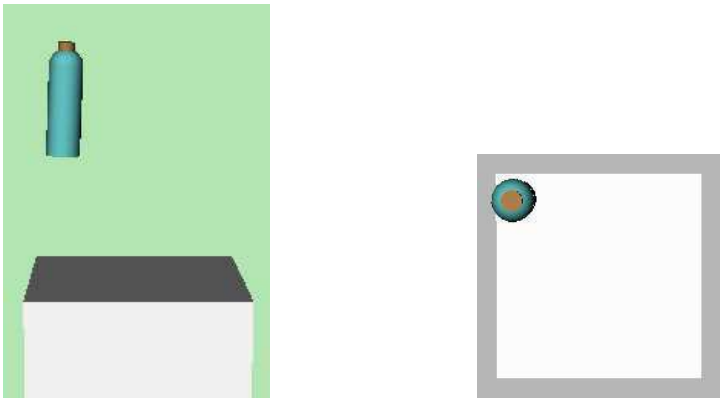
(a) Front view (b) Top view

Figure 7.48: Suggested initial state for experiment 6, based on skewed data (3000 instances).



(a) Front view (b) Top view

Figure 7.49: Estimated worst initial state for experiment 6, using uniform data (5000 instances).



(a) Front view (b) Top view

Figure 7.50: Estimated worst initial state for experiment 6, using skewed data (3000 instances).

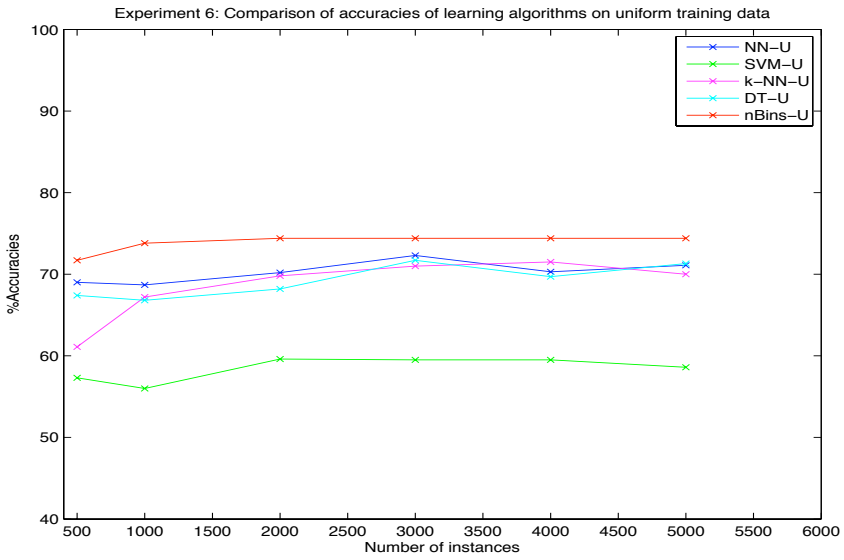


Figure 7.51: Comparison of accuracies of learning algorithms for experiment 6, with uniform training data.

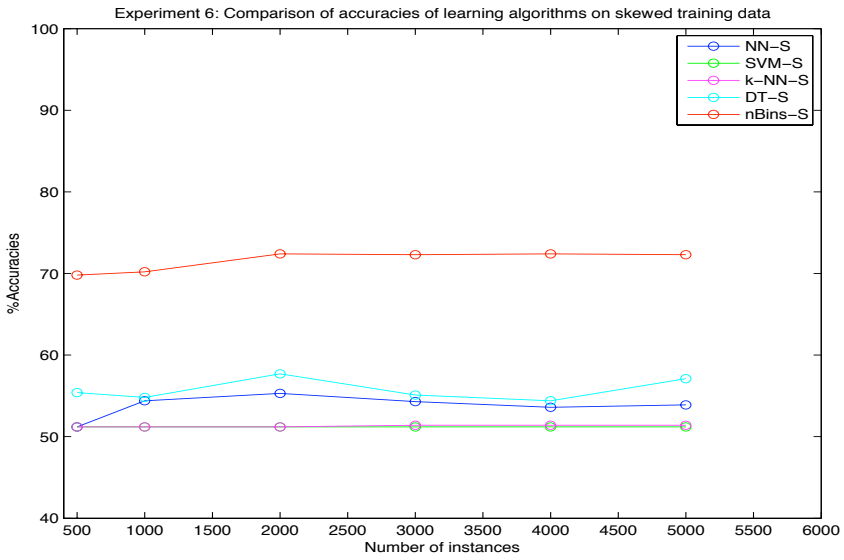


Figure 7.52: Comparison of accuracies of learning algorithms for experiment 6, with skewed training data.

7.7 Experiment 7

Specifications

In this experiment the simulation of the sample behavior shows that a `ball` is dropped `over` the center of a `basket` such that the `ball` stays `in`side the `basket` at the end of the simulation. Table 7.31 gives the specifications of the objects used in this experiment. The

Objects	**Dimensions** (m)	**Mass** (kg)
`Ball`	0.04(r)	0.3
`Basket`	0.4(h) $\times$ 0.3(l) $\times$ 0.24(w)	∞

Table 7.31: Specifications of the objects for experiment 7.

height (h) of the `basket` corresponds to the height of the sides.

Simulation description

The approach places 106 markers over the surface of the `ball` and finds following description of the simulation.

S_{init} $\equiv$ `Over(Ball, Basket)` $\wedge$ `Zz(Ball, Basket)` $\wedge$ `None(Ball)` $\wedge$
`Stationary-V(Ball)` $\wedge$ `Stationary-AV(Ball)`.

S_{final} $\equiv$ `In(Ball, Basket)` $\wedge$ `Zz(Ball, Basket)` $\wedge$ `None(Ball)` $\wedge$
`Stationary-V(Ball)` $\wedge$ `Stationary-AV(Ball)`.

Average time for a single simulation in this experiment is calculated to be 1.53 *sec*.

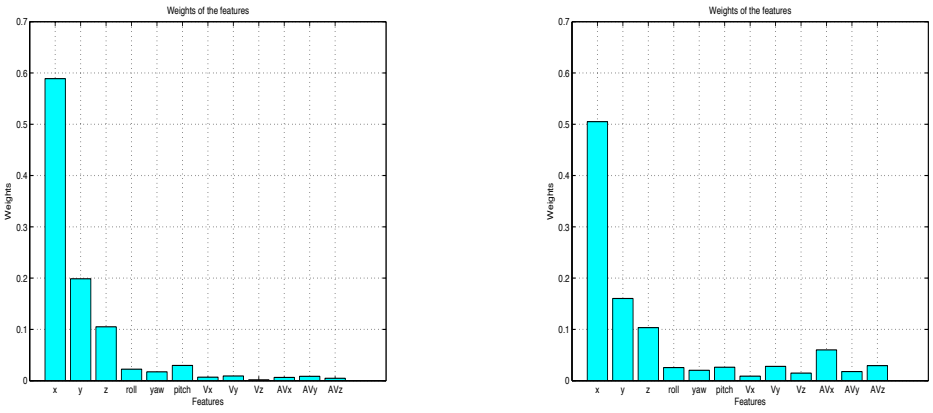
Limits of the parameters

For this experiment we change our usual practice of using the models of the objects obtained from the sample behavior in generating the instances of the action. Here, for the process of example generation we replace the model of `object-2` (i.e. the `basket`) with another model of the same object. The new model of the `basket` contains some arbitrary objects inside it. These arbitrary objects are assumed to be solid and static. Both the uniform and the skewed data sets of instances is generated using the new model of the `basket`. Figure 7.53 gives the comparison of the weights of the parameters calculated using these data sets. Limits of the parameters and distributions of the generated examples are given in table 7.32 and 7.33 respectively. From table 7.33:

$$P(success|refinedLimits) = \frac{3938}{5000} = 0.787$$

For this experiment $TSim_u = 7.02\ hrs$ (i.e. 25261 sec) and $TSim_s = 2.93\ hrs$ (i.e. 10563 sec), which makes:

$$TSim_u = 2.39 \times TSim_s$$



(a) 5000 instances (uniform data) (b) 3000 instances (skewed data)

Figure 7.53: Comparison of the weights of the parameters for two different sets of instances, for experiment 7.

Parameter(s) δ(s)	**Lower limit** Initial	Refined	**Upper limit** Initial	Refined
x	-0.2500	-0.1601	0.2500	0.1590
y	-0.1875	-0.1101	0.1875	0.1082
z	0.0650	0.0692	1.015	1.014
ρ	-0.7853	-0.7836	0.7853	0.7607
θ	-0.7853	-0.7844	0.7853	0.7846
ϕ	-0.7853	-0.7831	0.7853	0.7850
$\dot{x}, \dot{y}, \dot{z}, \dot{\rho}, \dot{\theta}, \dot{\phi}$	-0.0010	-0.0009	0.0010	0.0009

Table 7.32: Initial and refined limits of the parameters in Δ, for experiment 7.

Stage of simulation process	+ve examples	−ve examples	Extra evaluations
Transition	404	99	845
Termination	3938	1062	856

Table 7.33: States of the simulation process for experiment 7.

Modification of planning operator

Tables 7.34 and 7.35 give the details of the values of the parameters, suggested for the initial states of the objects. The tables also give the information regarding the bins used by N-Bins algorithm for this experiment. Suggested initial states of the objects based on uniform data and skewed data are shown in figure 7.54 and 7.55 respectively. From these

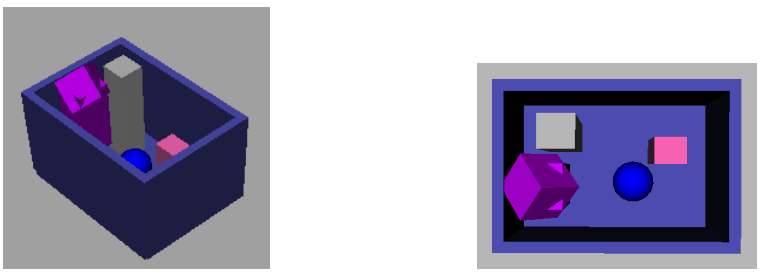
figures it is clear that N-Bins algorithm suggests that the safest way to drop a ball inside a basket is to release it inside the basket just over its bottom surface. The worst way to perform the same action, as estimated by N-Bins algorithm, is shown in figure 7.56 and 7.57. These two figures clearly show that N-Bins suggests that it is inappropriate for a `ball` to be dropped over other objects (from a maximum height) inside the `basket`. For this experiment, accuracies of the learning algorithms are shown in figure 7.58 and 7.59.

Parameter(s) δ(s)	**Suggested values**	**Bin limits** Lower	Upper	**Number of bins**
x	0.0349	0.0290	0.0408	27
y	-0.0301	-0.0374	-0.0228	15
z	0.1123	0.0693	0.1552	11
ρ	-0.4941	-0.5906	-0.3976	8
θ	0.6726	0.5605	0.7846	7
ϕ	-0.3475	-0.4347	-0.2604	9

Table 7.34: Selected values and bins of the parameters based on uniform data (5000 instances) for experiment 7.

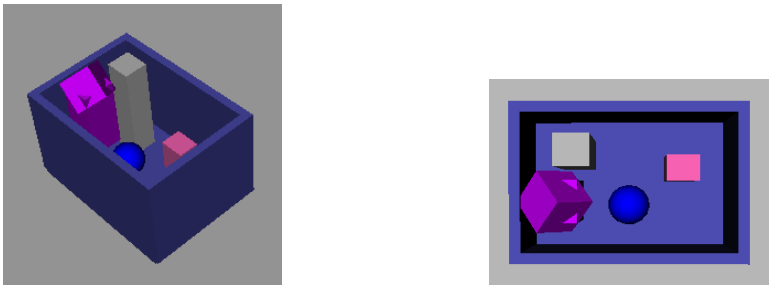
Parameter(s) δ(s)	**Suggested values**	**Bin limits** Lower	Upper	**Number of bins**
x	-0.0003	-0.0062	0.0056	27
y	-0.0433	-0.0506	-0.0360	15
z	0.1121	0.0691	0.1551	11
ρ	-0.3898	-0.5202	-0.2593	6
θ	0.6541	0.5234	0.7848	6
ϕ	-0.4481	-0.5601	-0.3360	7

Table 7.35: Selected values and bins of the parameters based on skewed data (3000 instances) for experiment 7.



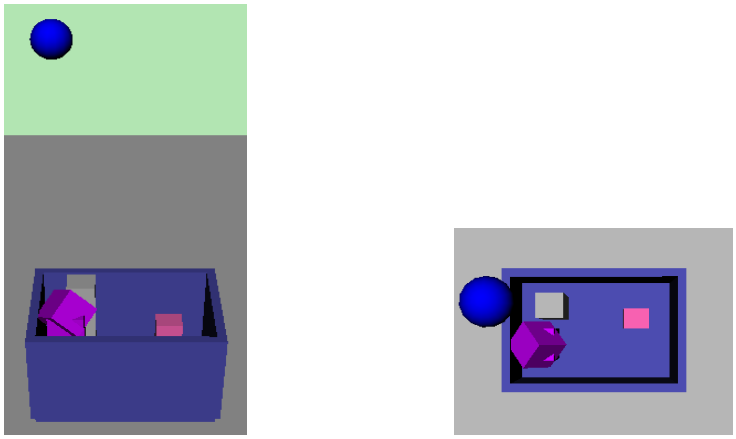
(a) Side view (b) Top view

Figure 7.54: Suggested initial state for experiment 7, based on uniform data (5000 instances).



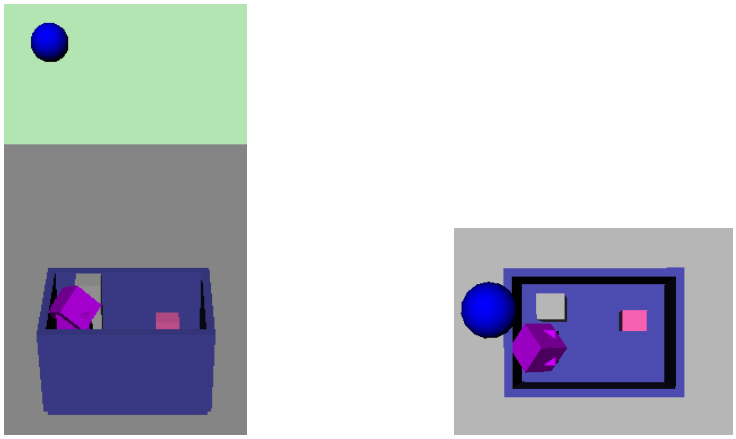
(a) Side view (b) Top view

Figure 7.55: Suggested initial state for experiment 7, based on skewed data (3000 instances).



(a) Front view (b) Top view

Figure 7.56: Estimated worst initial state for experiment 7, using uniform data (5000 instances).



(a) Front view (b) Top view

Figure 7.57: Estimated worst initial state for experiment 7, using uniform data (3000 instances).

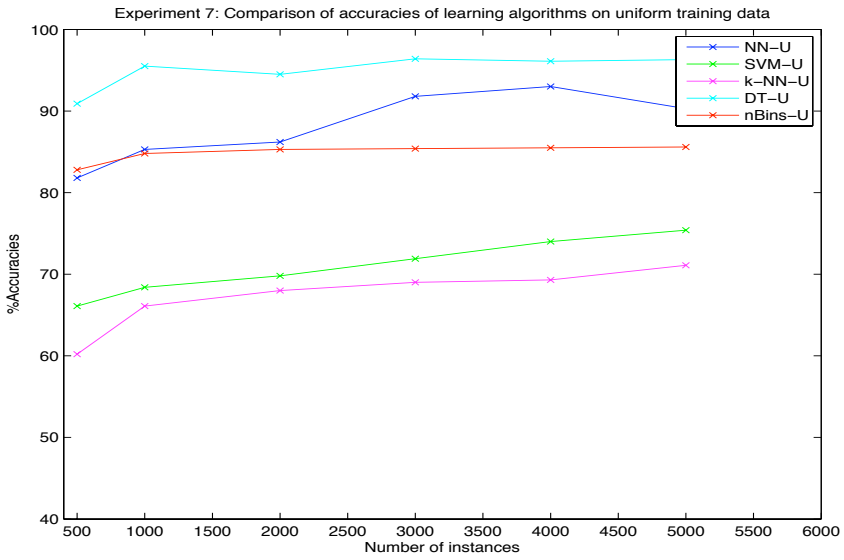


Figure 7.58: Comparison of accuracies of learning algorithms for experiment 7, with uniform training data.

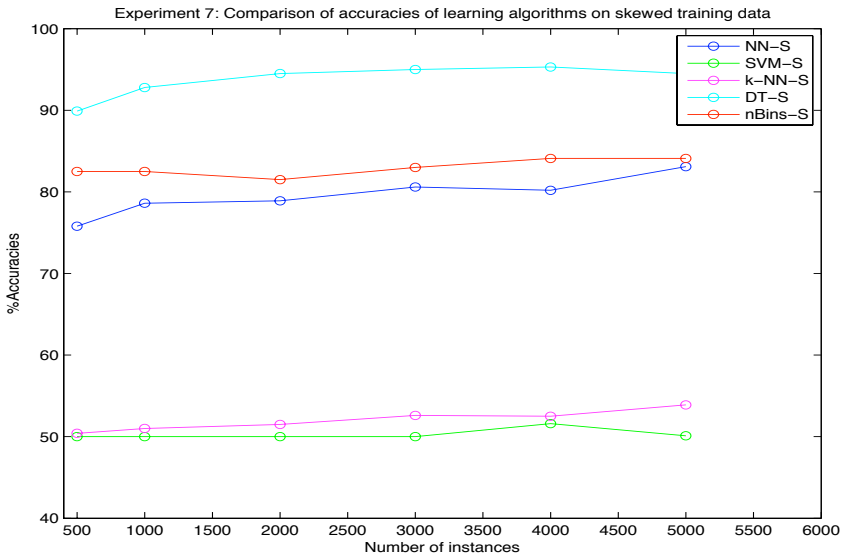


Figure 7.59: Comparison of accuracies of learning algorithms for experiment 7, with uniform training data.

7.8 Experiment 8

Specifications

In this experiment, the simulation of the sample behavior shows that a `ball` is *thrown towards* a `basket` such that in the final state of the simulation the `ball` stays `in`side the `basket`. For this experiment, the complete vector of the values of the parameters in Δ is given below:

$$\boldsymbol{\mu} = [-0.2, -0.23, 1.0, 0.0, 0.0, 0.0, 0.4, 0.5, 0.0, 0.0, 0.0, 0.0]$$

In the simulation of the sample behavior, the specifications of the objects are exactly the same as those in table 7.31. Although it is not very practical to consider the scenario of throwing objects in the basket from out side (instead of simply dropping them over it), but we still report this experiment to show the changes in the results of our approach for such actions.

Simulation description

The description of the simulation for this experiment is found to be as following:

$\mathtt{S}_{init} \equiv$ `Above(Ball, Basket)` $\wedge$ `Nn(Ball, Basket)` $\wedge$ `None(Ball)` $\wedge$
`[Moving-PosVx(Ball)` $\wedge$ `Moving-PosVy(Ball)` $\wedge$ `Stationary-Vz(Ball)]` $\wedge$
`Stationary-AV(Ball)`.

$\mathtt{S}_{final} \equiv$ `In(Ball, Basket)` $\wedge$ `Pp(Ball, Basket)` $\wedge$ `None(Ball)` $\wedge$
`Stationary-V(Ball)` $\wedge$ `Stationary-AV(Ball)`.

On average, it took 1.73 *sec* for a single simulation to execute in this experiment.

Limits of the parameters

Just as in experiment 7, for the process of example generation we replace the model of the `basket` with another model which contains static solid objects inside it. This model is shown in figure 7.60. As can be seen, now the `basket` contains one more object as compared to the `basket` used in experiment 7. With such a model of the `basket`, estimating the values of the parameters such that `ball` still ends up inside the `basket` in the final state, is a complex goal. However, to achieve this goal we do not make any special changes[8] to any part of the approach. For this experiment the weights of the parameters for the uniform and the skewed data are shown in figure 7.61. Tables 7.36 and 7.37 show the information regarding the limits of the parameters and distributions

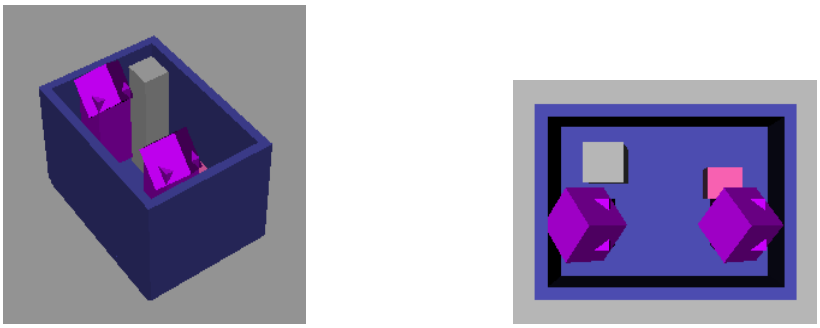
[8]We only replace 10% with 7% in line '3' of figure 5.9. This change is made to make the circumstances even harder for the approach.

of the examples for the example generation process. From table 7.37, we can calculate that:

$$P(success|refinedLimits) = \frac{952}{4048} = 0.19$$

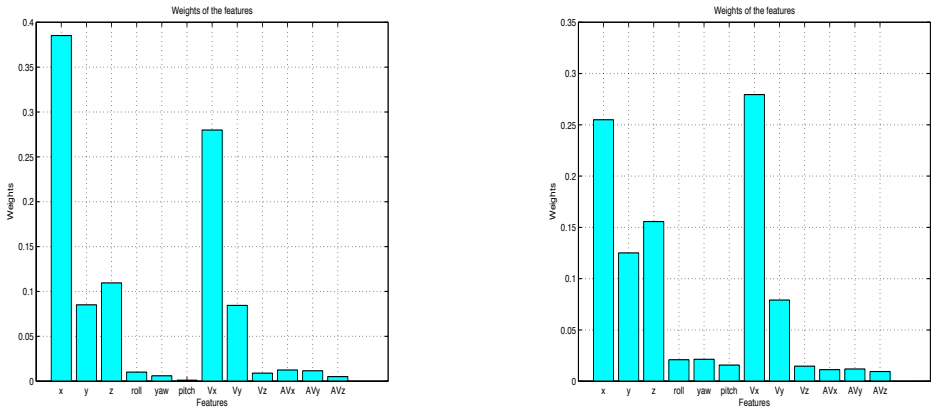
Furthermore, for this experiment $TSim_u = 11.96\ hrs$ (i.e. 43057 sec) and $TSim_s = 4.53$ hrs (i.e. 16336 sec), which makes:

$$TSim_u = 2.64 \times TSim_s$$



(a) Front view (b) Top view

Figure 7.60: Model of the `basket` used in the process of example generation.



(a) 5000 instances (uniform data) (b) 3000 instances (skewed data)

Figure 7.61: Comparison of the weights of the parameters for two different sets of instances, for experiment 8.

Modification of planning operator

Values of the parameters suggested by the N-Bins algorithm for this experiment are shown in table 7.38 and 7.39. Table 7.38 gives these values based on uniform data with 5000

Parameter(s) δ(s)	**Lower limit** Initial	Refined	**Upper limit** Initial	Refined
x	-0.2500	-0.2499	-0.0001	-0.2025
y	-0.1875	-0.1873	-0.0001	-0.0001
z	0.0650	0.0651	1.015	1.011
ρ	-0.7853	-0.7792	0.7853	0.7833
θ	-0.7853	-0.7693	0.7853	0.7820
ϕ	-0.7853	-0.7852	0.7853	0.7805
$\dot{x}$	0.001	0.0013	0.7500	0.7494
$\dot{y}$	0.001	-0.0039	0.7500	0.7473
$\dot{z}, \dot{\rho}, \dot{\theta}, \dot{\phi}$	-0.0010	-0.0009	0.0010	0.0009

Table 7.36: Initial and refined limits of the parameters in Δ, for experiment 8.

Stage of simulation process	+ve examples	−ve examples	Extra evaluations
Transition	76	275	7045
Termination	952	4048	21676

Table 7.37: States of the simulation process for experiment 8.

instances, whereas the values shown in table 7.39 are based on 3000 instances of skewed data. These tables also give the information regarding the number of bins and limits of the bins from which the suggested values are selected. Figures 7.62 and 7.63 show the frames of the motion of the `ball` when it is thrown at the `basket` according to the values of the parameters suggested in tables 7.38 and 7.39 respectively. These figures show that for both the uniform and the skewed data, the `ball` falls inside the `basket` at a place which is not occupied by other objects. In both the cases the `ball` slightly touches the wall of the `basket` during its projectile motion.

Parameter(s) δ(s)	**Suggested values**	**Bin limits** Lower	Upper	**Number of bins**
x	-0.2315	-0.2324	-0.2306	27
y	-0.1770	-0.1874	-0.1666	9
z	0.9684	0.9254	1.0114	11
ρ	-0.6490	-0.7793	-0.5188	6
θ	-0.3815	-0.5108	-0.2522	6
ϕ	-0.6287	-0.7853	-0.4721	5
$\dot{x}$	0.6747	0.6497	0.6996	15
$\dot{y}$	0.1434	0.0969	0.1898	8

Table 7.38: Selected values and bins of the parameters based on uniform data (5000 instances) for experiment 8.

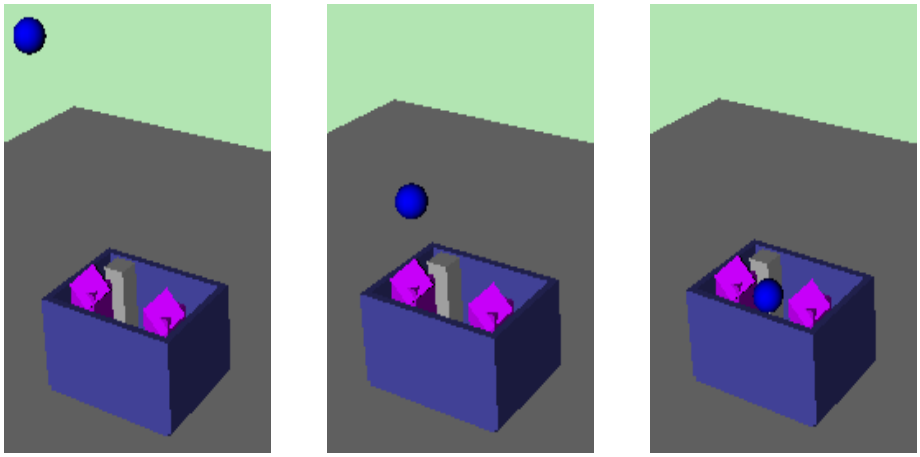
Parameter(s) δ(s)	**Suggested values**	**Bin limits** Lower	Upper	**Number of bins**
x	-0.2386	-0.2402	-0.2370	15
y	-0.1771	-0.1875	-0.1667	9
z	0.8845	0.8415	0.9274	11
ρ	-0.2244	-0.3360	-0.1127	7
θ	-0.2251	-0.3368	-0.1134	7
ϕ	0.1246	-0.0053	0.2545	6
$\dot{x}$	0.7355	0.7216	0.7493	27
$\dot{y}$	0.1470	0.1006	0.1933	8

Table 7.39: Selected values and bins of the parameters based on skewed data (3000 instances) for experiment 8.

Comparisons of the accuracies of the learning algorithms based on uniform and skewed data are shown in figure 7.64 and 7.65 respectively. In these figures, figure 7.65 is especially interesting. This figure shows that despite the fact that for this experiment $P(success|refinedLimits) = 0.19$, artificial neural networks outperforms N-Bins in accurately predicting the labels of test instances. Furthermore, N-Bins' accuracies start to decrease when the number of training instances are increased beyond 3000. According to our understanding, this phenomenon occurs because of the very strong dependence between the parameters x, y, z and $\dot{x}, \dot{y}, \dot{z}$. That is, if an object is required to be thrown at a target then in order to throw the object correctly, velocity of the object must always be selected according to its initial position. Such a strong dependence between the features of the instances is better captured by the hypothesis representation of neural networks[9] as compared to the bins of N-Bins algorithm. For N-Bins, increasing the number of training instances can cause deterioration in the performance because when the features are strongly dependent the patterns of the distributions of features (e.g. figure 7.2) start to extinguish with increasing number of training instances. This causes wrong selections of weights and finally wrong prediction of the labels of test instances.

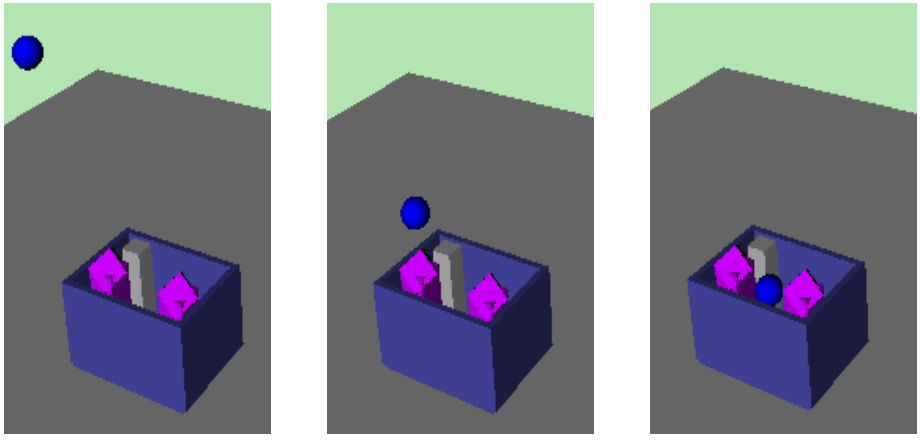
Although for this experiment performance of artificial neural networks is better than N-Bins, but it can be seen that for small number of instances of skewed data N-Bins' performance is also satisfactory. Since the main reason behind creation of the N-Bins algorithm is to achieve high accuracies with less amount of skewed data, we do not take any special measures against dependencies between the features in the algorithm.

[9]The thesis does not give any further discussion in regard of hypothesis representation of neural networks because it requires a detailed understanding of the learning principles of the algorithm. Interested readers can find details on the hypothesis representation of ANN in Mitchell [1997], chapter 4.



(a) Frame 1 (b) Frame 2 (c) Frame 3

Figure 7.62: Motion of the `ball` when it is thrown towards the `basket` with the values of the parameters suggested by N-Bins algorithms with the help of uniform data (5000 instances).



(a) Frame 1 (b) Frame 2 (c) Frame 3

Figure 7.63: Motion of the `ball` when it is thrown towards the `basket` with the values of the parameters suggested by N-Bins algorithms with the help of skewed data (3000 instances).

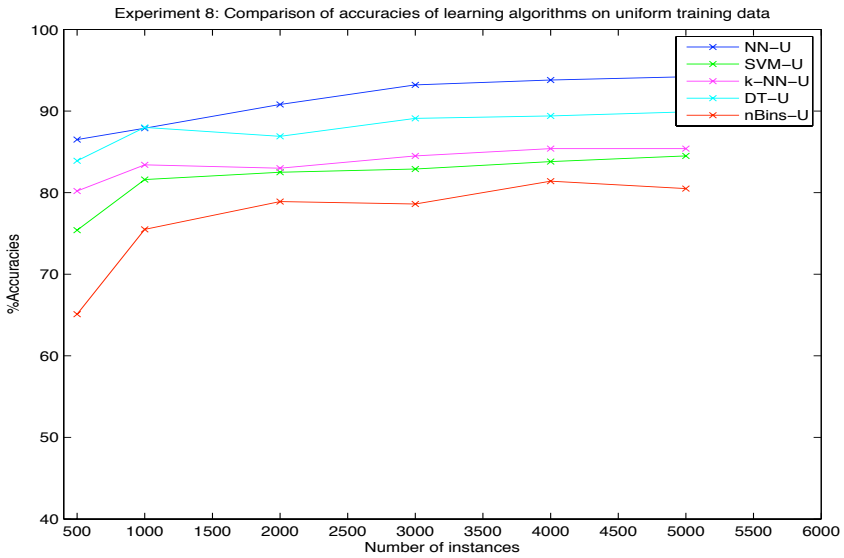


Figure 7.64: Comparison of accuracies of learning algorithms for experiment 8, with uniform training data.

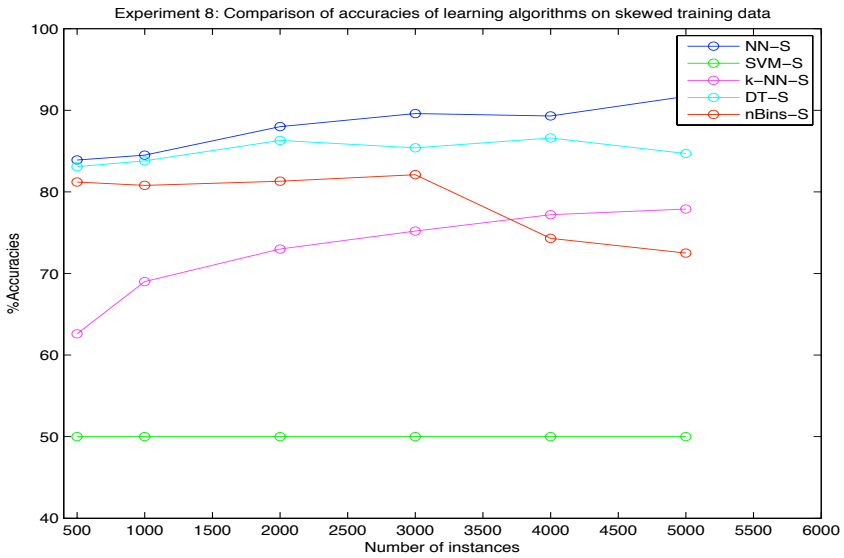


Figure 7.65: Comparison of accuracies of learning algorithms for experiment 8, with skewed training data.

7.9 Experiment 9

For this experiment we choose the same simulation for the sample behavior of the objects which is chosen in experiment 5 in section 7.5. However, in this experiment we change the model of the `table` in the process of generation of training examples. For the generation of examples, we use a model of the `table` that has static solid objects placed on it (see figure 7.67). Here, we do not give details about the *specifications* of the experiment and *simulation description* in order to avoid repetition. These details are same as those given in relevant description of section 7.5.

Limits of the parameters

The initial and the refined limits of the parameters for this experiment are also similar to those shown in table 7.22. Therefore, we do not show these limits here again. Table 7.40 shows the distribution of the examples according to the simulation process of this experiment. This distribution is very different from experiment 5 because of the presence of objects on the top surface of `object-2`. From the number of examples in the termination stage of the simulation process, we can calculate:

$$P(success|refinedLimits) = \frac{2911}{5000} = 0.582$$

Stage of simulation process	+ve examples	−ve examples	Extra evaluations
Transition	288	213	721
Termination	2911	2089	907

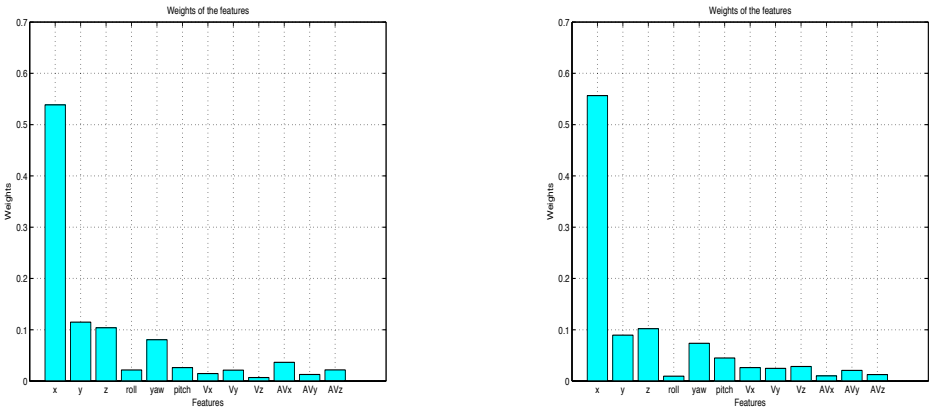
Table 7.40: States of the simulation process for experiment 9.

For this experiment $TSim_u = 4.1\ hrs$ (i.e. 14760 sec) and $TSim_s = 3.3\ hrs$ (i.e. 11930 sec), which makes:

$$TSim_u = 1.24 \times TSim_s$$

Modification of planning operator

Figure 7.66 shows the comparison of the weights of the parameters for the uniform and the skewed data sets of instances for this experiment. Information regarding the suggested values of the parameters and the bins is given in table 7.41 and 7.42 for both the data sets. For this experiment initial states of the objects with suggested values and estimated worst values of the parameters, based on both the data sets, are given in figure 7.67 - 7.70. Figure 7.71 and 7.72 give the comparisons of the accuracies of the learning algorithms for the uniform and the skewed training data for this experiment.



(a) 5000 instances (uniform data) (b) 3000 instances (skewed data)

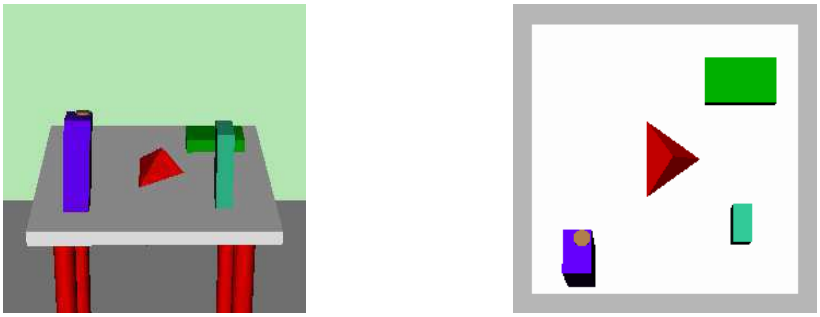
Figure 7.66: Comparison of the weights of the parameters for two different sets of instances, for experiment 9.

Parameter(s) δ(s)	**Suggested values**	**Bin limits** Lower	Upper	**Number of bins**
x	-0.1803	-0.1897	-0.1710	27
y	-0.2259	-0.2421	-0.2097	15
z	0.8567	0.8481	0.8653	11
ρ	-0.1631	-0.1957	-0.1305	6
θ	0.0431	0.0214	0.0648	9
ϕ	-0.0016	-0.1134	0.1102	7

Table 7.41: Selected values and bins of the parameters based on uniform data (5000 instances) for experiment 9.

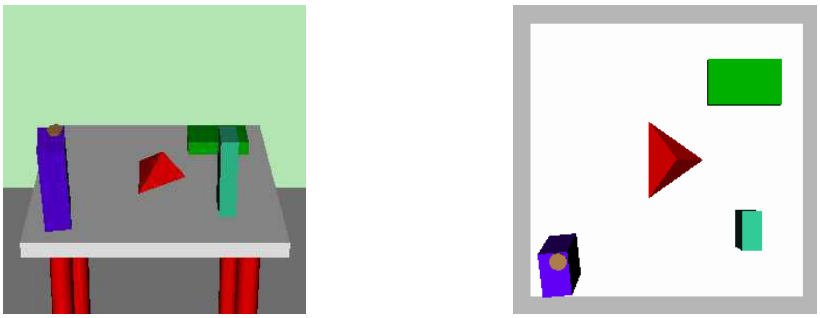
Parameter(s) δ(s)	**Suggested values**	**Bin limits** Lower	Upper	**Number of bins**
x	-0.2261	-0.2355	-0.2166	27
y	-0.2225	-0.2449	-0.2001	11
z	0.8375	0.8312	0.8437	15
ρ	0.1570	0.1177	0.1962	5
θ	-0.0440	-0.0658	-0.0223	9
ϕ	0.0951	-0.0025	0.1927	8

Table 7.42: Selected values and bins of the parameters based on skewed data (3000 instances) for experiment 9.



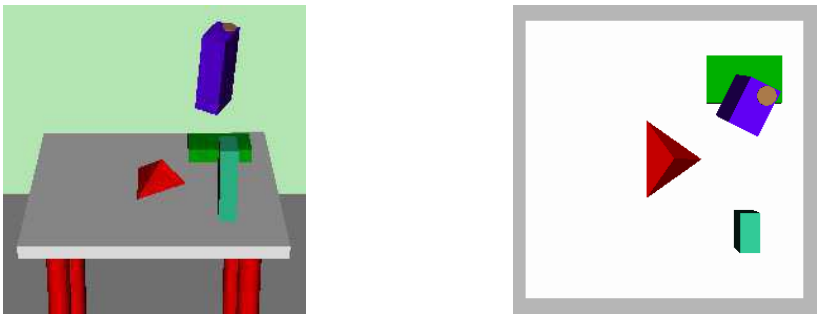
(a) Front view (b) Top view

Figure 7.67: Suggested initial state for experiment 9, based on uniform data (5000 instances).



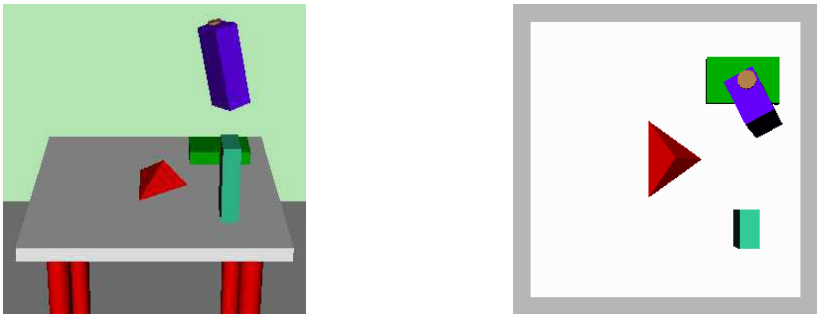
(a) Front view (b) Top view

Figure 7.68: Suggested initial state for experiment 9, based on skewed data (3000 instances).



(a) Front view (b) Top view

Figure 7.69: Estimated worst initial state for experiment 9, using uniform data (5000 instances).



(a) Front view (b) Top view

Figure 7.70: Estimated worst initial state for experiment 9, using skewed data (3000 instances).

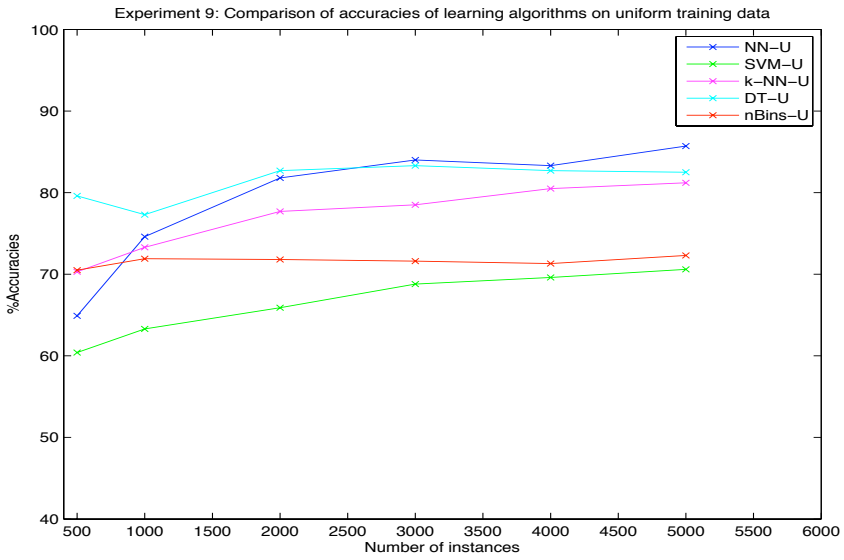


Figure 7.71: Comparison of accuracies of learning algorithms for experiment 9, with uniform training data.

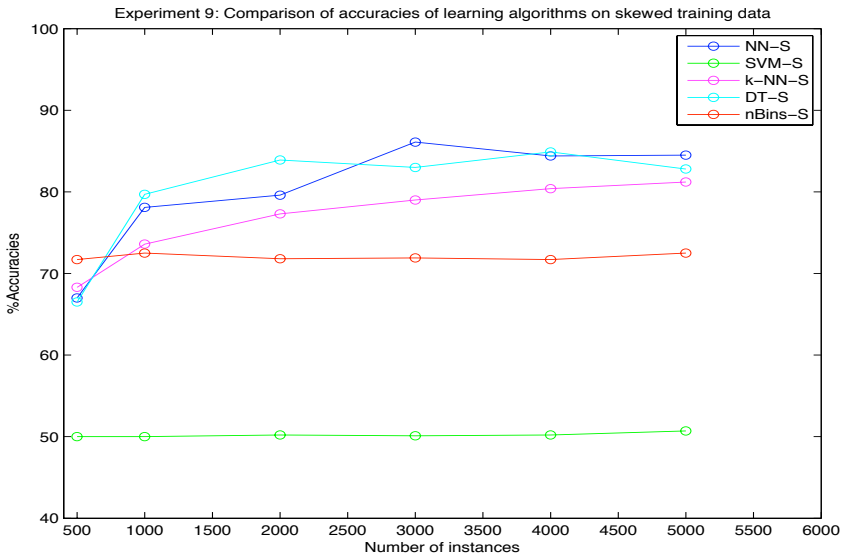


Figure 7.72: Comparison of accuracies of learning algorithms for experiment 9, with skewed training data.

7.10 Further analysis

In the experiments presented above, we show the results of applying our approach to the action of releasing an object over another object. We also show the results for an action where an object is thrown into another object. In our experiments we use different objects for better analysis of the approach. In the last three experiments (i.e. experiment 7 - 9) we also show the results by using the models of `object-2` other than those available from the sample simulation. In the explanation of the experiments, we also highlight and analyze some of the important aspects of the approach. However, we deliberately leave discussion on few aspects because their scope is more general. Below, we point out these aspects and give a brief explanation of each one of them:

- In the experiments presented in this chapter, *it appears that the description vocabulary is underutilized* and many of the details in chapter 6 are unnecessary for our work. However, this is not true. The predicates that appear in the experiments show only the simulation descriptions of the experiments. We also utilize the description vocabulary in labeling the examples. In all the experiments many of the predicates from the description vocabulary were evaluated in labeling the negative examples and also in the extra evaluations. Since in the experiments we only concentrated on the actions in which an object is released over (or thrown towards) another object, therefore the FOL expressions for the simulation descriptions are similar. This description is bound to change for other kinds of actions.

- *The process of example generation takes too much time* in all the experiments. The main reason behind this fact is that for generation of every single example, `object-1` has to 'fall' in the simulation. Although the time taken by each simulation in ODE (physics engine) can be minimized, but this minimization comes at the cost of accuracy of the simulation. Therefore, we do not utilize this option in our work. The time consumed by the simulation process can be greatly improved by running simulations in parallel. However, in our work we also do not take advantage of this option because of time constraints of the project.

- *In all the experiments* $TSim_u > TSim_s$. In the equation $TSim_u = n \times TSim_s$, average value of n is 3.8 for the experiments presented above. In our experiments, we need to generate the uniform data to obtain consistent high accuracies of the learning algorithms. We can see from the plots of the accuracies of the algorithms that generally the accuracies of the algorithms are not good for skewed data sets. However, in most of the experiments N-Bins is able to maintain a satisfactory accuracy even for the skewed data. Furthermore, it is visible from the figures in this chapter that N-Bins is able to suggest correct initial states of the objects with the

help of only 3000 skewed instances. Keeping in view this fact and accuracies of N-Bins and ANN, we can conclude that generally it is possible to consistently predict the behavior of the objects with reasonable accuracy with the help of only 3000 simulations. And, at the same time we can estimate the best and the worst possible initial states of the objects for an action. Based on the results of the experiments we propose a simple algorithm to accomplish this achievement. This algorithm is shown in figure 7.73. With the help of this algorithm it is possible to generate predictions with satisfactory accuracy while requiring very less time for the simulation process. For instance, for experiment 1 this algorithm can predict the behavior of the objects with 76.5% accuracy while requiring 3.7 hrs for the simulation process. Whereas, if we use the 5000 instance of uniform data we can achieve 82.5% accuracy in 52.7 hrs. We discuss this comparison between 3000 instances of skewed data and 5000 instance of uniform data because ML algorithms normally show better accuracies with more training instances. This means, we can expect an algorithm to show maximum accuracy using 5000 instances of uniform data.

1. Generate 3000 instances of the skewed data.
2. Estimate the best (and the worst) initial state(s) of the objects using N-Bins algorithm.
3. Train ANN with the available instances (using the specifications of ANN given in *appendix (D)*).
4. Predict the behavior of any initial state of the objects using N-Bins and ANN.
5. *if* both the algorithms predict the same behavior:
6. *then* consider the predictions to be correct.
7. *else*
8. *if* $0.3 < P(success|refinedLimits) < 0.7$:
9. *then* choose the prediction of ANN.
10. *else* choose the prediction of N-Bins.

Figure 7.73: Algorithm for using only 3000 skewed instances in initial state calculation and prediction of the behavior of objects for any initial state.

The algorithm shown in figure 7.73 is based on the results of the experiments shown in this chapter. Therefore, it has not been separately implemented.

- The accuracy plots in the experiments show that for some experiments, accuracies of learning algorithms other than ANN and N-Bins (e.g. D-Tree, SVM) are the best. However, the performance of these algorithms are not consistent in all the experiments. Therefore, it is not preferable to use these algorithms for the classifications of test instances.

- We do not mention (and analyze) the training and testing time for the learning algorithms in our experiments because this time is usually negligible as compared

to the time taken by the simulation process. However, it is worth mentioning that the training plus testing time[10] for (our implementation of) N-Bins algorithm is always less than 3 *sec* in each experiment (see *appendix (E)*).

In our experiments one of the major reasons of using different objects for each experiment is that we want to emphasize the point that our approach can be utilized to modify the planning operator of the robot without any special treatments for any particular object. Since we assume availability of the models of the objects, therefore in the process of example generation we always use the same model of `object-1` as the one used in the simulation of the sample behavior of the objects. However, it is also possible that at the time of creation of the sample behavior the model of `object-1` (and also `object-2`) is not known. We also conducted some primitive tests to extend our approach to such cases. We do this by creating the exact simulation descriptions and initial limits of the parameters shown in this chapter for experiment 1 - 6, by using only the objects of experiment 1 in the sample simulation. Since rest of the results of each of these experiments depend upon the simulation description and the initial limits of the parameters, therefore we can safely assume that rest of the results can also be successfully achieved. In the approach and the experiments presented in this thesis, we do not give any details about such extensions of our approach. Therefore, we also do not make any claim about such extension of the approach in this work. However it should be easy to see that the notion of finding the (logical) description of the sample simulation, provides an easy possibility to extend the proposed approach for the aforementioned cases.

As mentioned in the description of *modifying the planning operator* in section 5.2.4, the experiments presented above focus on the aspects of *conditions 1 - 3*, described in the said section. In these conditions, *condition 1* suggests that `object-1` should be released according to the values of the parameters suggested by the N-Bins algorithm. We can see in the figures of the initial states of the objects in all the experiments that selecting the suggested values (based on 3000 instances of skewed data) for the initial states always result in the desired behavior of the objects. *Condition 2* of the aforementioned conditions suggests that `obejct-1` should be released by selecting the values of the parameters randomly from the bins of the suggested values. It can be seen in the relevant tables of all the experiments that the widths of the bins of all the important parameters are very small. Because of this, it is always the case that in any initial state of the objects, created by selecting the values of the parameters randomly form the suggested bins, the values of the important parameters are very close to the mean of the bin limits. This makes the chances of achieving the desired behavior of the objects close to those of *condition 1* (i.e. 100%). To test this hypothesis, we perform 100 simulations of first 5 experiments in which we

[10]For 5000 training instances and 1000 test instances.

select the initial states of the objects according to *condition 2*. In all of these experiments the objects always achieved the desired behavior.

According to *condition 3* of the conditions mentioned in section 5.2.4, `object-1` should be released in an initial state that is predicted as a desired state (i.e. a positive example) by a classification algorithm with high accuracy. According to the accuracy plots of the learning algorithms in the experiments and the algorithm given in figure 7.73, the accuracy of correctly predicting an instance (with 3000 skewed instances) in our experiments ranges between $\sim$ 65 and $\sim$ 80%. It should be noticed that this range of accuracy is based on the assumption that there are equal chances of any initial state of the objects to be desirable or undesirable. That is, the accuracy of the algorithms are based on the uniform distributions of positive and negative examples in the test examples. In most of the experiments we see that $0.65 < P(success|refinedLimits) < 0.95$. Therefore, if a robot chooses an initial state of `object-1` from the refined limits then there is already a high chance of choosing a desired state. Hence, if an algorithm predicts an initial state chosen from the refined limits to be a positive instance then a robot can be more confident about the prediction of the algorithm than what is reflected in the accuracy of the algorithm.

From the discussion in this section and the results presented in the earlier sections of this chapter we can see that the approach presented in this work can enable a robot to estimate the (approximate) best way of performing an action that involves releasing of an object over another object. This way of performing the action can avoid occurrences of any unknown external faults because it is estimated by taking into account all the important parameters of the objects. The algorithms proposed in the work make it possible to estimate the best values of each of the parameters. The results show that this is accomplished by the proposed algorithms with very less number of training instances. The results also show that our work can enable a robot to estimate the (approximate) worst way of performing the action, within the refined limits of the parameters. A robot can also exploit the understanding of the 'worst way' in avoiding the occurrence of unknown external faults. The N-Bins algorithm also helps in predicting the behavior of the objects with reasonable accuracy where other algorithms fail to do so. In short, the result of the experiments presented in this chapter show that the proposed approach and the algorithms developed for this approach provide many possibilities of increasing the reliability of manipulation actions (that involve release of an object) against unknown external faults. All of these possibilities can be codified in the *conditions* of the body of `Allowed/3` predicate (shown in figure 5.1, chapter 5) for avoiding these faults.

8 Related Works

In this chapter we provide brief reviews of some of the works which are closely related to the problem handled in this thesis. Most of these reviews are about the works which deal with robotic faults. However, we also discuss some of the works which are not directly related to this area but the approaches used in these works either resemble our approach or foster understanding of our approach. Organization of this chapter is based on the areas to which the discussed works belong.

Robotic faults

Our previous work (Akhtar [2011]) is the most closely related work to the problem discussed in this thesis. In Akhtar [2011], we present an approach that uses naive physics to reason about unknown external faults in the area of robotics. In the said work we formulate the problem of unknown external faults and develop an approach that uses a qualitative version of physical laws on naive physics concepts to reason about the faults. Results of this work show a definite possibility of improvement in a robot's understanding of the situations that result in occurrences of unknown external faults. However, it is also found in the work that inherent limitations of qualitative abstraction of naive physics concepts and physical laws makes the process of formalization of knowledge very complex. The said work proposes and demonstrates the possibility of avoiding this complexity by categorizing the faults into different types. It proposes to formalize naive physics concepts as different logical frameworks suitable for particular types of faults. Although this approach helps in simplifying formalization of naive physics knowledge, but the large amount of knowledge required for obtaining reliable results from the approach remains a major limitation.

In general, Akhtar [2011] provides useful insights in dealing with unknown external faults. The approach developed in this thesis has made use of some of these insights without thoroughly discussing them. Whenever required, this thesis has referred to the said work for details. In both of our works we do not consider presence of external agents in the environment of robots and we assume natural physical phenomena as the only cause of external faults. Karg et al. [2011] takes the idea of external faults to further extents by considering behavior of the humans present in a robot's environment as the source of unknown external faults. Authors of this work propose to formalize common sense knowledge to develop understanding of *normality* in an environment where robots coexist with humans. This knowledge can enable a robot to detect unexpected events in the environment which remain unforeseen in the development phase of the system. Although

the said work is only a proposal, but we discuss it as a related work because the understanding of its authors about external faults is very similar to our understanding of these faults.

Ueda et al. [2011] presents a system architecture that is able to detect and recover from faults in real time. This work primarily deals with the external faults, where the faults are caused by external agents. The authors see the external faults as errors[1] which are detected by the robot at planning level through visual feed back. Such faults are termed as global errors by the authors. The architecture presented in the work, recovers from global errors by re-planning the tasks. The work also deals with the types of faults which occur at the lower level of abstraction (e.g. geometric level). These faults are termed as local errors. The architecture deals with such faults by continuously updating the geometric information about the environment and using the updated information in the execution of actions. This work detects global errors at symbolic level which is similar to our proposal of detecting the faults. However, main focus of this work is on recovering from the errors (with an implicit assumption that the robot knows how to do so) through re-planning, which is different from our objective.

In Ueda et al. [2011] authors illustrate working of their approach with an example where a robot is required to pick a bottle from a table and the bottle is displaced by a human during the execution of the action of the robot. The robot detects a local error there and keeps moving its manipulator to the bottle and finally picks it up. In the example of global error, the bottle is removed from the table by the human. The robot stops there and waits for the bottle to return `on` the table, then it executes the rest of the plan. It can be seen in these examples, that the robot is dealing with unexpected situations in its environment. However, the strategy of dealing with the situations assumes that the robot already knows what to do in such situations such that it can avoid failing its actions. This aspect of the work is very different from our approach because in our work we assume that a robot does not know how to avoid the failures and it is required to figure it out such that it can prevent the occurrence of the situations which cause the failures.

Manipulation planning

Although manipulation planning is not the main issue of discussion in this thesis, but some works in this area can be considered relevant to our approach. The main reason for this relevance is the commonality between the formulation of the planning problem expected by our approach and the actual formulation of the problem in the said works.

[1] We do not attach too much importance to the difference in the terms *faults* and *errors* here because these terms are subjective to formulation of the problem. Although we are convinced that even in Ueda et al. [2011]'s settings *faults* is also a very suitable term, however we use the term *errors* in the description in order to follow the convention of the original work.

For instance, in our approach we define a planning operator for the action of putting an object on another object and we expect that this operator represents only the action of release of one object over another. In plan based robotics such operators usually represent actions which are more than mere release of objects. For example in Okada et al. [2008b] a `Place` operator (i.e. an equivalent of `Put-on` operator) also represents the whole body reaching movement of the robot in addition to releasing the object. Defining planning operators in such a manner is not suitable for our approach. Works like Jason et al. [2010] define planning operators separately for releasing and picking up objects. Such definitions of planning operators are much more suitable for the application of our approach.

Dexterous manipulation and qualitative representation

Dexterous manipulation is an area in robotics in which multiple manipulators (e.g. manipulators or fingers) cooperate to grasp and manipulate objects (Okamura et al. [2000]). Most of the works in this area of robotics are concerned with anthropomorphic manipulators. The main objective of such works is to find suitable grips, manipulation movements and forces on objects for homogeneous and dexterous manipulation of objects. The distinguishing feature of the area of dexterous manipulation is that it is object-centered. That means, the problems in dexterous manipulation are formulated in terms of the objects which are to be manipulated Okamura et al. [2000]. Behavior of the manipulated objects and forces exerted on them are central issues of interest in dexterous manipulation. This makes this area relevant to our work. In our work, we primarily put constraints over the parameters of the manipulated objects. Although we are not interested in in-hand manipulation of objects, but it is possible that the constraints suggested by our approach can only be achieved through dexterous manipulation.

In the area of dexterous manipulation there are many works (e.g. Vinayavekhin [2009], Tegin et al. [2006]) which are dedicated to take advantage of human demonstration and learning process for better grasping of objects. The basic idea of the approaches in such works is similar to our approach. There is also a similar relevance between our approach and few works in qualitative representation and reasoning branch of AI. In our work we use relations (i.e. predicates) between the objects to capture their states. Generally speaking, such relations are used for reasoning purpose in QR literature. However, few works (e.g. Kohler et al. [2004]) also use these relations to store and filter the information of the states of the world. Which is similar to what we do in our approach.

In the above description of related works we indicate works in different areas and give a brief general review of such works, while mentioning their similarities to our approach. We do not discuss the approaches of these works in detail. The reason for that is, the *exact* approaches of these works are not very relevant to our approach because they do not deal with the problem of *unknown external faults*. This problem is not very well known

yet and this work is among the first few to take on this problem. Therefore, there are not many works which can be categorized as 'closely related' to this thesis. The approach developed in this thesis makes use of insights from different areas in a manner similar to the description of this chapter. It is believed that the works and areas discussed in this chapter can be useful in further developments in the area of *unknown external faults* in robotics.

9 Conclusion and future work

In this work an approach is presented to increase reliability of mobile manipulators against *unknown external faults*. This work focuses on increasing the reliability of the actions of manipulators which involve releasing of objects. The proposed approach assumes the settings of plan based robotic systems, in which a mobile manipulator is able to detect the occurrence of an unknown external fault with the help of its planning operators. Furthermore, it assumes availability of a sample simulation that shows the expected behavior of the manipulated object. This expected behavior is the one in which the manipulated object satisfies the *effects* of the planning operator that causes the detection of the fault. The sample simulation and the definition of the planning operator that detects the fault become the input to the proposed approach.

The proposed approach is formulated as a three-step scheme. In the first step of this scheme, a description of the sample simulation is found. This description is composed of logical sentences which describe the states of the objects in the simulation at the start and at the end of the simulation. The logical sentences are conjunctions of predicates which are used for capturing different aspects of the objects in the said states. These aspects include *connectedness* of the objects, *relative direction* of the objects; *orientation* of the manipulated object; *motion* of the manipulated object and *containment* of the objects. This work also formalizes the definitions of the predicates which cover all these aspects. Collectively, we refer to these predicates as *description vocabulary*. The second step of the scheme uses the description of the sample simulation to find the limits of the parameters of the manipulated object. These limits correspond to the extreme values of the parameters within which a robot can release the object and the behavior of the object can still remain the same (qualitatively) as its behavior shown in the sample simulation.

The second step of the scheme also uses the limits of the parameters to generate multiple examples of the behavior of the objects. These examples are generated by simulating the objects by selecting random values of the parameters of the simulation within the found limits. Based on the behavior of the objects in the simulation the examples are labelled as desired or undesired (i.e. positive or negative, or; class 1 or class 0). This step uses the description vocabulary and the description of the sample behavior to label the examples. The labelled examples become input for the third step of the scheme. This step finds the (approximate) best state of the manipulated object, in which if it is released then it can avoid the occurrences of unknown external faults. To accomplish the goal of the third step we propose an algorithm, that we refer as N-Bins. This algorithm suggests the

best state of the object by dividing the values of all of its parameters into different bins. Each parameter is divided into the bins according to its importance in the behavior of the manipulated object. The algorithm suggests the best state by selecting the mean values of the best bins of each parameter. The best bins are the bins that contain the values of the parameters that are strongly associated with the desired behavior of the objects in the simulation.

The main objective of the third step is to modify the definition of the planning operator received by the scheme as an input. This modification must help the robot to avoid the occurrence of the detected unknown external fault in the future. Therefore, it is done in this step by adding a new predicate in the *preconditions* of the planning operator. This predicate is defined to be true only under the *conditions* (regarding the releasing state of the object) which ensure that the action of release will not encounter unknown external faults. In this thesis we propose these conditions to be, that:

1. the releasing state of the object is the same as the best state estimated by the N-Bins algorithm, or
2. the releasing state of the object is formed by the values of the parameters (of manipulated object) chosen from the best bins found by N-Bins algorithm, or
3. the releasing state is predicted to be a desired state by N-Bins and/or ANN algorithms.

The above mentioned conditions vary in their level of strictness. Furthermore, whether or not a robot can satisfy each one of these conditions, is subjective to the circumstances in which the robot performs the action. Therefore, we show the results of our experiments by showing the results related to each of the above conditions separately. In this work we also do not commit to an absolute definition of the predicate added in the *preconditions* of a modified planning operator. This is because, we see possibility of many useful definitions for this predicate. Therefore, the proposal of the modification of the planning operator (in the third step) is kept only theoretical in this work.

The results of experiments with the approach show that using the proposed approach a robot can be enabled to release the objects in a manner that avoids the occurrence of unknown external faults. The proposed approach also enables a robot to explicate the worst way of performing the action. This understanding can also be exploited in avoiding the unknown external faults. The results of the experiments also show that the N-Bins algorithm proposed in this work also enables a robot to predict the behavior of the objects with reasonable consistency and accuracy[1]. This task is accomplished by the algorithms presented in this work with only 3000 simulations.

[1]If it is used along with ANN in a manner shown in figure 7.73.

Future work

Successful results of the application of the approach encourages further exploration of many aspects of this work. Some of these are stated below:

- *Improvement in N-Bins algorithm* is a promising aspect. Because of time constraints of the project the algorithm has not been *very* thoroughly investigated. It is believed that, keeping the basic theory of the algorithm intact, it can be improved to show even better accuracy of prediction. Since the algorithm uses only basic mathematical calculations and does not require iterative learning process, better accuracy from the algorithm can make it especially valuable in terms of computational efficiency.
- *Application of the basic scheme to other actions/tasks of a robot* is another interesting aspect. Broadly speaking, one of the main idea behind the approach proposed in this work can be seen as following: "Give a robot an example of an action to be executed and let the robot find the best way to do it, by itself". The robot finds the best way by considering all parameters involved in the action, such that the values of the parameters found by the robot ensure that the action is completed without occurrence of unknown external fault. The main idea and the techniques used for materializing it seemingly have vast applications. This work provides the basic ingredients (e.g. description vocabulary, N-Bins) for such applications. Therefore, extending and applying the proposed approach to other actions of a robot is a promising future direction.
- *Parallelization of the process of example generation* can greatly improve the time required by the approach to produce the desired results. The time taken by each simulation is a major limitation of using a simulator in creating training examples. Since we already concentrate on achieving useful results with smaller number of training examples, further reduction in time required by the simulation process can greatly benefit the practical application of the approach.

10 Appendix

A: Definitions

Below are some important definitions (found in the literature regarding fault diagnosis) of the terms used in the thesis.

Error
A deviation between a measured value of an output variable and its true or theoretically correct value.

Failure
A permanent interruption of a system's ability to perform a required function under specified operating conditions.

Fault
An unpermitted deviation of at least one characteristic property of parameter of the system from the acceptable/usual/standart condition.

Fault detection
Determination of the presence of fault in the system and the time of detection.

Fault isolation
Determination of kind, location and time of detection of a fault.

Fault identification
Determination of the size and time-variant behavior of a fault.

Fault diagnosis
Determination of the kind, size, location and time of detection of the fault.

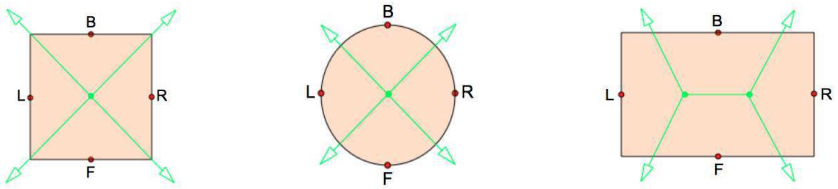
Jointly Exhaustive and Pairwise Disjoint (JEPD)
Relations between two regions are JEPD if at any time there is one and exactly one relation that holds between the regions.

Malfunctioning
An intermittent irregularity in the fulfillment of a system's desired function.

Mean time between failures
Predicted elapsed time between inherent failures of a system during operation.

B: Geometric calculations

Figure 10.3 shows the geometries of the surfaces of `object-2` considered in this work. The red points on the boundaries of the shapes show the position of the markers for each surface. The line segments (shown as green rays) correspond to voronoi decompositions of the shapes according to the markers. The green points are the voronoi nodes for the decompositions.



(a) Square (b) Circle (c) Rectangle

(d) Semicircle + rectangle (e) Two semicircles + rectangle

Figure 10.1: Geometric shapes for the top surfaces of `object-2` with voronoi decomposition.

Calculations for $l_{\mathcal{M}}$ and $f_{\mathcal{M}}$

Square:

Let the width of the square be x. Since the rays are (and always will be) passing through the corners of the square, therefore:

$$l_{\mathcal{M}} = x \quad \text{for all the markers} \tag{10.1}$$

Since for all the markers $l_{\mathcal{M}}$ is equal to one fourth of the length of the perimeter of the square, therefore:

$$f_{\mathcal{M}} = \frac{1}{4} \quad \text{for all the markers} \tag{10.2}$$

Circle:

Let the radius of the circle be r. This makes the circumference of the circle $2\pi r$. Since the rays are (and always will be) dividing the circumference of the circle into four arcs of equal lengths, therefore:

$$l_{\mathcal{M}} = \frac{2\pi r}{4} = \frac{\pi r}{2} \quad \text{for all the markers} \tag{10.3}$$

and

$$f_{\mathcal{M}} = \frac{1}{4} \quad \text{for all the markers} \tag{10.4}$$

Rectangle:

Let the width and height of the rectangle be $2x$ and $2y$ respectively. To calculate $l_{\mathcal{M}}$ and $p_{\mathcal{M}}$, let us consider only the lower left corner of the rectangle. This corner is shown in figure 10.2 as a triangle made by joining the markers L and F by the line segment $\overline{LF}$. In this triangle,

$$|\overline{LF}| = \sqrt{x^2 + y^2}; \quad \cos\theta = \frac{x}{|\overline{LF}|} \quad \text{and} \quad |\overline{VF}| = \frac{|\overline{LF}|}{2}$$

Furthermore,

$$\cos\theta = \frac{|\overline{VF}|}{|\overline{WF}|}$$

From these facts we have,

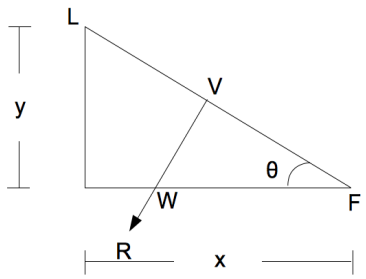


Figure 10.2: Lower left corner of the rectangle in figure 10.3(c).

$$\frac{x}{\sqrt{x^2 + y^2}} = \frac{\sqrt{x^2 + y^2}}{2|\overline{WF}|} \quad \text{and}$$
$$2|\overline{WF}| = \frac{x^2 + y^2}{x}$$

From figure 10.3 we can see that $2|\overline{WF}| = l_{\mathcal{M}}$ for the marker F. Hence;

$$l_{\mathcal{M}} = \frac{x^2 + y^2}{x} \quad \text{for markers } F \text{ and } B \tag{10.5}$$

Since the perimeter of the rectangle is $4(x + y)$, therefore to find $l_{\mathcal{M}}$ for R and L we subtract $4(x + y)$ from $2|\overline{WF}|$. After simplification;

$$l_{\mathcal{M}} = 2y(1 + y) \quad \text{for markers } R \text{ and } L \tag{10.6}$$

Using the fact that $f_{\mathcal{M}}$ is the fraction of the perimeter corresponding to $l_{\mathcal{M}}$, we have;

$$f_{\mathcal{M}} = \frac{x^2 + y^2}{4x(x + y)} \quad \text{for markers } F \text{ and } B \tag{10.7}$$

$$f_{\mathcal{M}} = \frac{2y(1 + y)}{4(x + y)} \quad \text{for markers } R \text{ and } L \tag{10.8}$$

Semicricle + rectangle:

Let the width and height of the rectangle involved in the shape be x and y. Let the radius of the semicircle be r. In this case $l_{\mathcal{M}}$ for the marker L can be given by equation 10.6. It can be seen from figure 10.3(d) that if we assume figure 10.2 to be representing the case of figure 10.3(d) then for B and F, $l_{\mathcal{M}}$ is the sum of $|\overline{WF}|$ and $\frac{x}{2}$. Thus;

$$l_{\mathcal{M}} = \frac{2x^2 + y^2}{2x} \quad \text{for makers } F \text{ and } B \tag{10.9}$$

For R, $l_{\mathcal{M}}$ can be given by equation 10.3. Whereas,

$$l_{\mathcal{M}} = \frac{2\pi r}{8} + \frac{x}{2} = \frac{1}{4}(\pi r + 2x) \quad \text{for makers } N1 \text{ and } N2 \tag{10.10}$$

The perimeter of the shape in figure 10.3(d) is the sum of three sides of a rectangle and half of the circumference of a circle. With $p = 4x + 2y + \pi r$, $f_{\mathcal{M}}$ for each marker can be given as following:

$$f_{\mathcal{M}} = \frac{2y}{p}(1 + y) \quad \text{for marker } L \tag{10.11}$$

$$f_{\mathcal{M}} = \frac{\pi r}{2p} \quad \text{for marker } R \tag{10.12}$$

$$f_{\mathcal{M}} = \frac{2x^2 + y^2}{2xp} \quad \text{for makers } F \text{ and } B \tag{10.13}$$

$$f_{\mathcal{M}} = \frac{1}{4p}(\pi r + 2x) \quad \text{for makers } N1 \text{ and } N2 \tag{10.14}$$

Two semicircles + rectangle:

Let the width and height of the rectangle involved in the shape to be x and y respectively. Let the radius of each semicircle be r. With these notations, $l_{\mathcal{M}}$ for markers R and L can be given by equation 10.3. For $N1, N2, N3$ and $N4$, $l_{\mathcal{M}}$ can be given by equation 10.10. For F and B,

$$l_{\mathcal{M}} = x \tag{10.15}$$

As can be seen in figure 10.3(e) that the perimeter of the shape is same as twice the width of the rectangle, plus the circumference of the circle with radius r. With $p = 4x + 2\pi r$, $f_{\mathcal{M}}$ for R and L can be given by equation 10.13. For $N1, N2, N3$ and $N4$ it can be given by equation 10.14 and for F and B,

$$l_{\mathcal{M}} = \frac{x}{p} \tag{10.16}$$

C: Shapes of the object used in testing connectedness relations

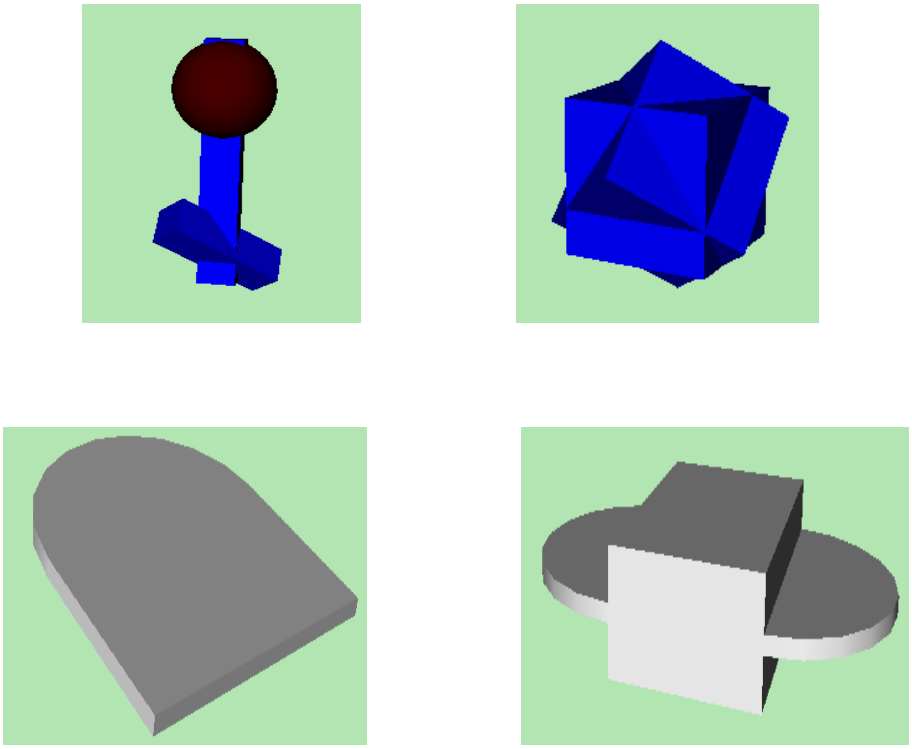


Figure 10.3: Examples of geometric shapes used to evaluate connectedness predicates.

D: Specifications of learning algorithms

Following are the specification of the learning algorithms used in the work.

Artificial Neural Networks:

Parameter	Value
Learning rate	0.05
Momentum	0.2
Number of hidden layers	1
Activation function	Sigmoid

Support Vector Machines:

Parameter	Value
Kernel	Polynomial
Degree	2
C	0.0

K-Nearest Neighbors:

Parameter	Value
K	20
Weighted votes	On

Decision Trees:

Parameter	Value
Criterion	Information gain
Minimal gain	0.1
Maximal depth	20
Minimal split size	4

N-Bins

Parameter	Value
binMul	2
augBin	3

E: Accuracies of learning algorithms

In this appendix we give the numerical values of the accuracies of the learning algorithms used in our experiments in chapter 7. The accuracies are given in tabular form. Each table gives the accuracies for a single learning algorithm for the uniform and skewed data sets. When the accuracies of prediction of some algorithm is no better than 50% for a particular training data set, then the table does not show the accuracies for that data set (e.g. table 10.2). Each table also gives the information regarding the precision and recall of the algorithms. The time for the learning process is also shown. Except for N-Bins, all the accuracies and learning times are for the implementations of the algorithms in RapidMiner software. N-Bns algorithm is implemented separately in Matlab. The time reported in each table also includes the time taken by the algorithm to predict the labels of 1000 test instances.

Experiment 1

Neural Network					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	82.5	85.2	78.6	29
	4000	83.1	86.4	78.6	24
	3000	80.7	85.1	74.4	17
	2000	76.5	78.4	73	11
	1000	66.8	67.1	65.8	6
	500	63.3	64.3	59.6	3
Skewed	5000	65.9	59.7	97.4	23
	4000	54.7	52.5	99.8	19
	3000	63.5	57.9	98.6	15
	2000	55.5	52.9	99.0	9
	1000	56.1	53.3	99.0	5
	500	56.1	53.3	99.2	3

Table 10.1: Accuracies of neural network for experiment 1

Support Vector Machine					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	83.0	87.8	76.6	12
	4000	82.7	87.4	76.4	10
	3000	82.4	87.3	75.8	3
	2000	79.3	84.3	72	2
	1000	74.3	76.8	69.6	1
	500	71.9	73.6	68.2	1

Table 10.2: Accuracies of SVM for experiment 1

k-Nearest Neighbours					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	78	81.8	72	2
	4000	78.1	81	73.4	2
	3000	75	76.7	71.8	2
	2000	71.1	71.1	71.2	2
	1000	69.7	70.3	68.2	1
	500	68.2	69.2	65.6	1

Table 10.3: Accuracies of k-NN for experiment 1

Decision tree					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	80.4	83.5	75.8	9
	4000	78.8	82.7	72.8	7
	3000	77.1	79.4	73.2	5
	2000	73.9	77.6	67.2	4
	1000	69.7	70.9	66.8	3
	500	70.8	76.5	60	1
Skewed	5000	69.7	63.2	94.2	4
	4000	69.9	63	96.2	3
	3000	69.2	62.7	94.8	2
	2000	65.6	59.6	96.6	2
	1000	66.6	60.8	93.6	2
	500	64	58.7	94.6	2

Table 10.4: Accuracies of D-tree for experiment 1

N-Bins					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	73.7	82.8	59	2
	4000	74.3	84	60	2
	3000	72.8	82.9	57.4	2
	2000	69.1	75.8	56	2
	1000	69.5	77.1	55.4	2
	500	68.6	76.7	53.4	2
Skewed	5000	75	71.4	83.4	2
	4000	75.1	71.6	83	2
	3000	76.5	74.2	81.2	2
	2000	70.2	66.5	81.2	2
	1000	68.1	65.1	78	2
	500	66.5	62.8	81.2	2

Table 10.5: Accuracies of N-Bins for experiment 1

Neural Network					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	82.9	85.6	79.0	29
	4000	83.5	88.1	77.4	24
	3000	80.0	84.4	73.6	14
	2000	78.0	80.8	73.4	9
	1000	68.6	69.2	66.8	4
	500	67.4	69.2	62.6	3
Skewed	5000	65.4	59.3	98.2	23
	4000	54.5	52.4	98.6	19
	3000	63.1	57.7	97.6	15
	2000	54.9	52.6	98.2	9
	1000	55.5	53.0	97.8	5
	500	56.0	53.3	97.6	3

Table 10.6: Accuracies of neural network for experiment 1 (10 % noise)

Support Vector Machine					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	82.6	86.9	76.8	12
	4000	82.4	86.8	76.4	10
	3000	81.3	85.0	76.0	3
	2000	79.4	83.1	73.8	2
	1000	75.1	78.2	69.6	1
	500	71.2	71.7	70	1

Table 10.7: Accuracies of SVM for experiment 1 (10 % noise)

k-Nearest Neighbours					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	79.9	84.7	73.0	2
	4000	77.7	81.1	72.2	2
	3000	76.2	78.7	71.8	2
	2000	73.5	75	70.4	2
	1000	71.5	72.9	68.4	1
	500	71.6	73.5	67.6	1

Table 10.8: Accuracies of k-NN for experiment 1 (10 % noise)

Decision tree					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	80.4	82.6	77	9
	4000	79.1	81.8	74.8	7
	3000	78.7	78.4	78.9	5
	2000	73.9	75.9	70.0	4
	1000	71.0	71.6	69.4	3
	500	73.3	78.1	64.8	1
Skewed	5000	68.3	62.1	94.0	4
	4000	68.4	62.1	94.6	3
	3000	69.7	63	95.4	2
	2000	64.5	59.0	95.2	2
	1000	66.8	60.7	95	2
	500	60.7	56.6	91.2	2

Table 10.9: Accuracies of D-tree for experiment 1 (10 % noise)

N-Bins					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	76.6	87.4	62.2	2
	4000	77.0	87.0	63.4	2
	3000	75.5	86.3	60.6	2
	2000	72	81.6	56.8	2
	1000	72.2	81.5	57.4	2
	500	70.2	78.5	55.6	2
Skewed	5000	74.5	71.0	84.4	2
	4000	73.6	69.6	83.6	2
	3000	75.6	72.6	82.2	2
	2000	71.0	67.7	78	2
	1000	68.9	65.8	78.8	2
	500	68.3	64.1	83.2	2

Table 10.10: Accuracies of N-Bins for experiment 1(10 % noise)

Experiment 2

Neural Network					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	86.2	86.6	85.9	27
	4000	83.7	83.0	84.8	19
	3000	84.4	84.0	85.0	18
	2000	81.3	80.4	82.8	12
	1000	81.0	82.9	81.2	6
	500	76.5	74.4	80.8	2
Skewed	5000	84.1	81.0	89.0	29
	4000	84.3	82.2	87.6	18
	3000	80.6	76.9	87.4	16
	2000	79.9	88.6	75.5	11
	1000	76.5	72.8	84.6	6
	500	68.5	64.1	84.2	3

Table 10.11: Accuracies of neural network for experiment 2

Support Vector Machine					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	83.6	81.8	86.4	89
	4000	83.6	82.2	85.8	8
	3000	83.3	82.6	84.4	7
	2000	82.4	81.6	83.6	3
	1000	79.3	76.0	85.6	3
	500	75.9	73.2	81.8	2
Skewed	5000	77.0	70.6	92.4	42
	4000	78.3	72.7	90	5
	3000	76.3	69.7	93.2	5
	2000	74.7	68	93.4	2
	1000	71.7	65.2	93.2	2
	500	66.1	60.5	92.4	2

Table 10.12: Accuracies of SVM for experiment 2

k-Nearest Neighbours					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	75.4	77.1	72.2	2
	4000	73.9	76.6	68.8	2
	3000	73.2	76.5	67.0	2
	2000	71.7	75.0	65.0	2
	1000	70.8	66.8	82.8	1
	500	69.6	66.1	80.6	1
Skewed	5000	70.3	64.8	89	3
	4000	72.4	67.9	85	2
	3000	68.4	63.5	86.8	2
	2000	70.0	65.0	86.8	1
	1000	66.8	61.7	88.6	1
	500	62.4	58.3	87.4	1

Table 10.13: Accuracies of k-NN for experiment 2

Decision tree					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	79.4	78.8	80.4	14
	4000	78.5	77.5	80.2	7
	3000	78.3	77.2	80.4	6
	2000	73.9	72.8	76.4	2
	1000	72.6	72.8	72.2	1
	500	73.3	74.3	71.2	1
Skewed	5000	74.9	71.6	82.6	14
	4000	76.8	72.7	85.8	7
	3000	73.9	70.7	83.6	2
	2000	73.8	70.2	82.6	2
	1000	71.5	67.5	82.8	2
	500	67.7	64.3	79.4	1

Table 10.14: Accuracies of D-tree for experiment 2

N-Bins					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	75.0	77.6	70	2
	4000	74.1	76.5	69.6	2
	3000	74.2	76.6	69.6	2
	2000	73.4	75.3	69.6	2
	1000	74.3	77.8	68.0	2
	500	70.8	74.5	63.2	2
Skewed	5000	72.9	70.7	78.2	2
	4000	73.8	72.3	77.2	2
	3000	72.3	70.3	77.2	2
	2000	72.4	70.4	77.2	2
	1000	71.7	69.5	77.2	2
	500	69.3	66.9	76.2	2

Table 10.15: Accuracies of N-Bins for experiment 2

Neural Network					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	85.6	86.6	84.2	27
	4000	83.1	86.0	79.0	19
	3000	85.1	85.3	84.8	18
	2000	81.6	82.5	80.2	12
	1000	79.8	79.6	80.2	6
	500	76.8	78.0	74.6	2
Skewed	5000	82.9	87.3	77.0	29
	4000	82.5	86.9	76.6	18
	3000	80.2	84	74.6	16
	2000	79.1	87.0	68.4	11
	1000	77.2	83.7	67.6	6
	500	69.8	76.6	57.0	3

Table 10.16: Accuracies of neural network for experiment 2 (10 % noise)

Support Vector Machine					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	82.6	84.7	79.6	90
	4000	82.7	84.7	79.8	8
	3000	81.8	83.1	79.8	7
	2000	80.5	81.6	78.8	3
	1000	78.0	81.5	72.4	4
	500	75.7	78.6	70.6	2
Skewed	5000	78.8	89.6	65.2	42
	4000	78.4	87.4	66.4	5
	3000	76.6	90.3	59.6	3
	2000	74.6	90.2	55.2	2
	1000	72.9	89.4	52.0	2
	500	69.0	88.0	44.0	2

Table 10.17: Accuracies of SVM for experiment 2 (10 % noise)

k-Nearest Neighbours					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	76.9	74.7	81.4	2
	4000	76.2	74.2	80.4	2
	3000	75.6	73.6	79.8	2
	2000	74.0	72.2	78.0	2
	1000	70.2	77.0	57.6	1
	500	70.8	75.2	62.0	1
Skewed	5000	68.9	81.8	48.6	3
	4000	71.9	79.8	58.6	2
	3000	67.3	77.6	48.6	2
	2000	68.0	79.0	49	1
	1000	64.7	77.7	41.2	1
	500	64.3	79.2	38.8	1

Table 10.18: Accuracies of k-NN for experiment 2 (10 % noise)

Decision tree					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	77.9	78.9	76.2	14
	4000	78.0	78.3	77.4	7
	3000	78.3	78.2	78.4	6
	2000	74.6	74.8	74.2	2
	1000	72.0	72.7	70.4	1
	500	72.8	72.3	73.8	1
Skewed	5000	75.2	80.3	66.8	14
	4000	76.2	80.9	68.6	7
	3000	74.8	80.2	65.8	4
	2000	71.5	77.1	71.5	2
	1000	70.0	75.5	59.2	2
	500	68.7	74.2	57.4	1

Table 10.19: Accuracies of D-tree for experiment 2 (10 % noise)

N-Bins					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	75.8	77.8	72.2	2
	4000	75.9	77.9	72.2	2
	3000	76.7	78.5	73.4	2
	2000	75.5	76.8	73.0	2
	1000	74.7	76.2	71.8	2
	500	74.0	77.9	67.0	2
Skewed	5000	72.2	68.8	81.0	2
	4000	73.4	71.5	77.8	2
	3000	70.9	67.5	80.4	2
	2000	70.5	67.0	80.8	2
	1000	70.5	67.1	80.4	2
	500	68.1	65.2	77.6	2

Table 10.20: Accuracies of N-Bins for experiment 2(10 % noise)

Experiment 3

Neural Network					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	88.6	90.9	85.8	29
	4000	87.3	86.2	88.8	21
	3000	83.7	80.8	88.4	18
	2000	85.2	89.5	79.8	12
	1000	84.9	86.3	83.0	6
	500	73.8	73.7	74.0	3
Skewed	5000	56.2	53.4	99.4	28
	4000	56.8	53.7	98.8	20
	3000	55.9	53.2	98.4	17
	2000	56.3	53.4	98.8	11
	1000	53.9	52	99.6	5
	500	51.9	50.8	99.4	3

Table 10.21: Accuracies of neural network for experiment 3

Support Vector Machine					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	88.2	93.2	82.4	27
	4000	87.7	92.7	81.8	6
	3000	87.4	91.5	82.4	3
	2000	86.8	91.8	80.8	2
	1000	83.5	85.1	81.2	2
	500	80.2	80.9	79.0	2

Table 10.22: Accuracies of SVM for experiment 3

k-Nearest Neighbours					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	81.5	86.9	74.2	2
	4000	82.2	86.3	76.6	2
	3000	81.7	85.8	76.0	1
	2000	77.7	81.8	71.2	1
	1000	74.2	76.5	69.8	1
	500	69.3	70.9	65.4	1

Table 10.23: Accuracies of k-NN for experiment 3

Decision tree					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	83.6	83.9	83.2	10
	4000	81.2	81.0	81.0	6
	3000	82.3	81.2	84.0	3
	2000	82.6	82.4	82.8	2
	1000	79.9	79.0	81.4	2
	500	77.4	77.5	77.2	1
Skewed	5000	66.4	60.2	96.6	6
	4000	66.0	60.0	95.4	4
	3000	66.5	60.3	96.2	3
	2000	62.5	57.5	96.0	2
	1000	60.6	56.3	95.0	2
	500	55.8	53.2	97.0	1

Table 10.24: Accuracies of D-tree for experiment 3

N-Bins					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	80.1	85.5	72.4	2
	4000	79.3	84.1	72.2	2
	3000	77.8	81.7	71.6	2
	2000	77.2	80.9	71.2	2
	1000	76.0	79.4	70.2	2
	500	68.7	71.7	61.8	2
Skewed	5000	66.0	60.5	92.0	2
	4000	67.2	61.3	93.2	2
	3000	65.7	60.0	94.8	2
	2000	61.7	57.1	94.0	2
	1000	57.7	54.3	95.4	2
	500	56.3	53.8	88.8	2

Table 10.25: Accuracies of N-Bins for experiment 3

Experiment 4

Neural Network					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	92.7	95.3	89.9	25
	4000	90.4	91.9	88.6	18
	3000	90.9	94.5	86.8	13
	2000	89.4	90.2	88.4	9
	1000	88.0	89.4	86.2	4
	500	83.1	85.3	80	3
Skewed	5000	74.6	67	97	26
	4000	75.7	67.7	98.4	18
	3000	67.2	60.8	97	13
	2000	67.2	60.6	98.0	9
	1000	66.2	60	97.0	4
	500	67.3	60.9	96.6	3

Table 10.26: Accuracies of neural network for experiment 4

Support Vector Machine					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	79.6	75.9	86.0	20
	4000	78.3	73.8	87.8	13
	3000	77.8	73.3	87.4	8
	2000	75.1	70.1	87.4	4
	1000	74.9	69.8	87.8	2
	500	71.3	66.3	86.6	2

Table 10.27: Accuracies of SVM for experiment 4

k-Nearest Neighbours					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	81.4	88.1	72.6	2
	4000	81.3	87.9	72.6	2
	3000	80.2	86.6	71.4	1
	2000	79.4	86.6	69.9	1
	1000	77	82.9	68.0	1
	500	76.1	81.3	67.8	1

Table 10.28: Accuracies of k-NN for experiment 4

Decision tree					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	87.1	88.1	85.8	4
	4000	87.3	88.4	85.8	3
	3000	87.0	88.7	84.8	1
	2000	86.4	87.1	85.4	1
	1000	83.5	84.8	81.6	1
	500	83.1	84.8	80.6	1
Skewed	5000	74.4	66.7	97.4	2
	4000	73.2	65.7	97.2	2
	3000	73.3	66.1	95.8	1
	2000	72.7	65.4	96.2	1
	1000	75.4	68.1	95.6	1
	500	55.8	53.2	97.0	1

Table 10.29: Accuracies of D-tree for experiment 4

N-Bins					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	74.2	83.2	60.6	2
	4000	74.9	83.5	62.0	2
	3000	75.0	83.9	61.8	2
	2000	73.7	82.6	60.0	2
	1000	72.8	81.3	59.2	2
	500	73.8	82.7	60.2	2
Skewed	5000	76.1	81.1	68.0	2
	4000	76.5	80.9	69.4	2
	3000	76.7	80.8	70.0	2
	2000	74.5	79.5	66	2
	1000	75.9	79.4	70.0	2
	500	75.4	76.2	73.8	2

Table 10.30: Accuracies of N-Bins for experiment 4

Experiment 5

Neural Network					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	78.2	75.8	82.4	23
	4000	78.0	75.8	81.9	18
	3000	78.3	76.1	82.3	13
	2000	77.7	75.8	81.2	8
	1000	69.4	70.6	66.2	4
	500	68.4	68.4	67.8	2
Skewed	5000	57.3	54	96.5	21
	4000	50.2	50.0	100	17
	3000	57.0	53.8	96.2	13
	2000	49.0	49.4	97.2	9
	1000	50.1	50	99.3	4
	500	50.3	50.1	100	2

Table 10.31: Accuracies of neural network for experiment 5

Support Vector Machine					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	71.7	73.7	67.1	13
	4000	71.3	73.4	66.5	6
	3000	72.1	75.0	65.8	6
	2000	70.4	72.7	64.9	8
	1000	65.1	66.9	59.2	3
	500	61.8	62.9	56.5	1

Table 10.32: Accuracies of SVM for experiment 5

k-Nearest Neighbours					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	73.5	76.8	67.1	2
	4000	72.8	75.4	67.3	2
	3000	71.5	73.4	67.1	1
	2000	68.3	68.8	66.2	1
	1000	65.0	65.0	64.7	1
	500	62.1	61.3	64.9	1

Table 10.33: Accuracies of k-NN for experiment 5

Decision tree					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	74.8	73.2	77.9	8
	4000	72.6	70.5	77.3	4
	3000	74.6	72.1	79.9	3
	2000	74.6	72.8	78.2	2
	1000	73.4	72.1	76	2
	500	74.3	72.3	78.6	1
Skewed	5000	61.8	57.3	91.7	7
	4000	61.6	56.8	94.9	3
	3000	60.6	56.2	94.0	2
	2000	48.7	49.2	94.1	1
	1000	59.4	55.5	93.2	1
	500	59.0	55.4	91.2	1

Table 10.34: Accuracies of D-tree for experiment 5

N-Bins					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	76.1	83.2	65.6	2
	4000	75.4	81.8	65.0	2
	3000	76.0	83.4	64.5	2
	2000	76.0	84.4	63.6	2
	1000	75.2	80.3	66.5	2
	500	76.3	83.4	65.4	2
Skewed	5000	75.7	78.8	70.0	2
	4000	77.3	81.5	70.4	2
	3000	75.5	78.4	70.2	2
	2000	72.5	73.8	69.4	2
	1000	72.5	73.8	69.5	2
	500	73.7	74.0	72.7	2

Table 10.35: Accuracies of N-Bins for experiment 5

Experiment 6

Neural Network					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	71.1	75.4	64.8	23
	4000	70.3	73.2	66.3	18
	3000	72.3	78.8	62.9	13
	2000	70.2	74.0	64.6	9
	1000	68.7	70.9	65.9	4
	500	69.0	71.5	65.6	2
Skewed	5000	53.9	52.7	98.1	23
	4000	53.6	52.5	97.3	18
	3000	54.3	52.9	95.4	13
	2000	55.3	53.6	95.4	9
	1000	54.4	53.0	95.6	4
	500	51.2	51.2	100	2

Table 10.36: Accuracies of neural network for experiment 6

Support Vector Machine					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	58.6	63.0	46.5	53
	4000	59.5	64.5	46.7	8
	3000	59.5	64.3	47.1	4
	2000	59.6	65.0	45.8	6
	1000	56.0	58.9	46.5	2
	500	57.3	60.3	48.8	2

Table 10.37: Accuracies of SVM for experiment 6

k-Nearest Neighbours					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	70.0	74.5	63.1	3
	4000	71.5	76.8	63.6	2
	3000	71.0	76.34	63.1	1
	2000	69.8	74.4	62.7	1
	1000	67.2	71.3	60.2	1
	500	61.1	63.6	56.3	1

Table 10.38: Accuracies of k-NN for experiment 6

Decision tree					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	71.3	80.0	58.6	4
	4000	69.7	78.7	56.1	2
	3000	71.7	80.6	59.0	2
	2000	68.2	69.9	67.3	1
	1000	66.8	66.6	70.6	1
	500	67.4	66.9	71.9	1
Skewed	5000	57.1	54.9	92.3	7
	4000	54.4	53.0	96.6	2
	3000	55.1	53.4	96.5	1
	2000	57.7	55.4	90.4	1
	1000	54.8	53.4	92.1	1
	500	55.4	53.6	95.9	1

Table 10.39: Accuracies of D-tree for experiment 6

N-Bins					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	74.4	82.5	63.4	2
	4000	74.4	82.5	63.4	2
	3000	74.4	82.5	63.5	2
	2000	74.4	82.5	63.5	2
	1000	73.8	84.1	60.2	2
	500	71.7	74.6	67.9	2
Skewed	5000	72.3	73.3	72.3	2
	4000	72.4	73.6	71.9	2
	3000	72.3	73.3	72.3	2
	2000	72.4	73.7	71.7	2
	1000	70.2	69.7	74.2	2
	500	69.9	68.9	75.0	2

Table 10.40: Accuracies of N-Bins for experiment 6

Experiment 7

Neural Network					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	90.3	91.7	88.6	29
	4000	93.0	92.7	93.4	22
	3000	91.8	91.1	92.6	17
	2000	86.2	85.9	86.6	11
	1000	85.3	86.7	83.4	5
	500	81.8	83.5	79.2	2
Skewed	5000	83.1	76.2	96.2	29
	4000	80.2	73.4	94.8	22
	3000	80.6	73.8	94.8	16
	2000	78.9	72.1	94.2	9
	1000	78.6	72.1	93.4	5
	500	75.8	69.2	93.0	2

Table 10.41: Accuracies of neural network for experiment 7

Support Vector Machine					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	75.4	71.7	83.8	23
	4000	74.0	70.0	83.0	14
	3000	71.9	68.3	81.8	7
	2000	69.8	66.5	79.8	3
	1000	68.4	64.8	80.4	1
	500	66.1	64	73.6	1

Table 10.42: Accuracies of SVM for experiment 7

k-Nearest Neighbours					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	71.1	71.9	69.2	2
	4000	69.3	70.3	66.8	2
	3000	69.0	69.4	68.0	1
	2000	68.0	68.8	65.8	1
	1000	66.1	67.3	62.6	1
	500	60.2	61.0	56.6	1

Table 10.43: Accuracies of k-NN for experiment 7

Decision tree					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	96.3	96.0	96.6	2
	4000	96.1	95.8	96.4	2
	3000	96.4	97.9	94.8	2
	2000	94.5	94.7	94.2	1
	1000	95.5	95.8	95.2	1
	500	90.9	92.1	89.4	1
Skewed	5000	94.5	92.4	97.0	2
	4000	95.3	93.4	97.4	2
	3000	95.0	92.6	97.8	1
	2000	94.5	91.6	98.0	1
	1000	92.8	89.0	96.8	1
	500	89.9	85.2	96.6	1

Table 10.44: Accuracies of D-tree for experiment 7

N-Bins					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	85.6	92.4	77.6	2
	4000	85.5	92.8	77.0	2
	3000	85.4	92.7	76.8	2
	2000	85.3	91.9	77.4	2
	1000	84.8	93.5	74.8	2
	500	82.8	87.8	76.2	2
Skewed	5000	84.1	84.7	83.2	2
	4000	84.1	85.9	81.6	2
	3000	83.0	84.5	80.0	2
	2000	81.5	81.7	81.4	2
	1000	82.5	85.4	78.4	2
	500	82.5	85.4	78.4	2

Table 10.45: Accuracies of N-Bins for experiment 7

Experiment 8

Neural Network					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	94.2	92.5	96.2	25
	4000	93.8	91.5	96.6	20
	3000	93.2	90.7	96.2	15
	2000	90.8	89.8	92.0	14
	1000	87.9	86.8	89.4	5
	500	86.5	85.8	87.4	3
Skewed	5000	91.7	96.8	86.2	23
	4000	89.3	97.6	80.6	18
	3000	89.6	95.0	83.6	14
	2000	88.0	95.0	80.2	11
	1000	84.5	93.2	74.4	5
	500	83.9	94.2	72.2	2

Table 10.46: Accuracies of neural network for experiment 8

Support Vector Machine					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	84.5	88.2	79.6	23
	4000	83.8	88.2	78.0	14
	3000	82.9	87.5	76.8	8
	2000	82.5	88.6	74.6	6
	1000	81.6	87.2	74.0	1
	500	75.4	84.5	62.2	1

Table 10.47: Accuracies of SVM for experiment 8

k-Nearest Neighbours					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	85.4	80.6	93.2	2
	4000	85.4	80.6	93.2	2
	3000	84.5	79.6	92.8	1
	2000	83.0	77.8	92.4	1
	1000	83.4	77.6	94.0	1
	500	80.2	73.7	94.0	1
Skewed	5000	77.9	93.7	59.8	2
	4000	77.2	93.9	58.2	2
	3000	75.2	92.6	54.8	1
	2000	73.0	93.6	49.4	1
	1000	69.0	92.0	41.6	1
	500	62.6	92.0	27.6	1

Table 10.48: Accuracies of k-NN for experiment 8

Decision tree					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	89.9	89.2	90.8	5
	4000	89.4	89.6	89.2	3
	3000	89.1	89.7	88.4	2
	2000	86.9	86.8	87.0	1
	1000	88.0	89.2	86.4	1
	500	83.9	83.8	84.0	1
Skewed	5000	84.7	93.9	74.2	3
	4000	86.6	94.2	78.0	2
	3000	85.4	93.8	75.8	1
	2000	86.3	93.5	77.6	1
	1000	83.8	92.5	73.6	1
	500	83.1	89.3	75.2	1

Table 10.49: Accuracies of D-tree for experiment 8

N-Bins					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	80.5	76.8	87.4	2
	4000	81.4	77.2	89.0	2
	3000	78.6	77.0	81.6	2
	2000	78.9	75.9	84.6	2
	1000	75.5	76.3	74.0	2
	500	65.1	65.9	62.6	2
Skewed	5000	72.5	83.0	56.6	2
	4000	74.3	83.1	61.0	2
	3000	82.1	84.2	79.0	2
	2000	81.3	78.2	80.7	2
	1000	80.8	79.6	82.8	2
	500	81.2	80.7	82.0	2

Table 10.50: Accuracies of N-Bins for experiment 8

Experiment 9

Neural Network					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	85.7	83.0	89.8	24
	4000	83.3	80.4	88.0	19
	3000	84.0	81.9	87.2	14
	2000	81.8	81.3	82.6	9
	1000	74.6	74.7	74.4	4
	500	64.9	65.1	64.2	2
Skewed	5000	84.5	80.4	91.2	24
	4000	84.4	78.9	94	20
	3000	86.1	80.3	95.6	15
	2000	79.6	78.1	82.2	9
	1000	78.1	75.4	83.4	4
	500	67.0	65.2	72.8	2

Table 10.51: Accuracies of neural network for experiment 9

Support Vector Machine					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	70.6	71.8	67.8	25
	4000	69.6	70.3	67.8	14
	3000	68.8	68.4	69.8	8
	2000	65.9	64.4	71.2	4
	1000	63.3	61.2	71.6	3
	500	60.4	57.6	78.8	2

Table 10.52: Accuracies of SVM for experiment 9

k-Nearest Neighbours					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	81.2	83.7	77.0	2
	4000	80.5	82.1	78.0	2
	3000	78.5	80.5	75.2	1
	2000	77.7	80.5	75.2	1
	1000	73.3	74.1	71.6	1
	500	70.3	70.9	68.8	1
Skewed	5000	81.2	79.4	85.6	2
	4000	80.4	83.6	85.2	2
	3000	79.0	82.0	83.8	1
	2000	77.3	74.9	82.0	1
	1000	73.6	70.3	81.6	1
	500	68.3	65.0	79.2	1

Table 10.53: Accuracies of k-NN for experiment 9

Decision tree					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	82.5	83.0	81.8	6
	4000	82.7	83.4	81.6	4
	3000	83.3	83.4	83.2	2
	2000	82.7	83.9	81.0	2
	1000	77.3	77.6	76.8	1
	500	79.6	79.6	79.6	1
Skewed	5000	82.8	80.6	86.4	6
	4000	84.9	81.9	89.6	4
	3000	83.0	80.8	86.6	3
	2000	83.9	82.4	86.2	2
	1000	79.7	78.4	82.0	1
	500	66.5	64.9	71.8	1

Table 10.54: Accuracies of D-tree for experiment 9

N-Bins					
Data	# of training instances	Accuracy(%)	Precision(%)	Recall(%)	Time(s)
Uniform	5000	72.3	78.5	61.4	2
	4000	71.3	76.7	61.2	2
	3000	71.6	77.9	60.2	2
	2000	71.8	76.8	62.4	2
	1000	71.9	76.6	63.0	2
	500	70.5	75.7	60.4	2
Skewed	5000	72.5	80.3	59.6	2
	4000	71.7	77.4	61.2	2
	3000	71.9	78.3	60.6	2
	2000	71.8	79.0	59.4	2
	1000	72.5	78.3	62.2	2
	500	71.7	75.0	65.0	2

Table 10.55: Accuracies of N-Bins for experiment 9

CD Content

- This document as PDF
- Implementation of the approach
- A document containing title, abstract, supervisors' names and student name.
- Bibtex entry of the work.

Bibliography

Naveed Akhtar. Fault reasoning based on naïve physics. Technical report, University of Applied Science Bonn-Rhein-Sieg, February 2011. URL `http://opus.bib.hochschule-bonn-rhein-sieg.de/opus-3.3/volltexte/2011/5/`. WS09/10 H-BRS Supervised by Prof. Paul G. Plöger, Ms. Küstenmacher. 12, 15, 23, 88, 128

Naveed Akhtar and Anastassia Kuestenmacher. Using naive physics for unknown external faults in robotics. In *22nd International Workshop on Principles of Diagnosis (DX-2011)*, 2011. 1, 23

Ivan Bratko. *PROLOG Programming for Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 2001. ISBN 0201416069. 7

Jennifer Carlson and Robin R. Murphy. Reliability analysis of mobile robots. In *ICRA*, pages 274–281, 2003. 14

A. G. Cohn and S. M. Hazarika. Qualitative spatial representation and reasoning: An overview. *Fundam. Inf.*, 46(1-2):1–29, 2001. 10, 11

Matthew J. Daigle. *A qualitative event-based approach to fault diagnosis of hybrid systems.* PhD thesis, Graduate School of Vanderbilt University, 2008. 15

J de Kleer and B C Williams. Diagnosing multiple faults. *Artif. Intell.*, 32(1):97–130, 1987. ISSN 0004-3702. 14

Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010. 22

Kenneth D. Forbus. *Qualitative Reasoning*. MIT Press, 2003. 10

Kenneth D. Forbus, Paul Nielsen, and Boi Faltings. Qualitative spatial reasoning: The clock project. *Artif. Intell.*, 51(1-3):417–471, 1991. 10

Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning theory and practice*. Morgan Kaufmann Publishers Inc., 2004. 7, 10

L. Honghai and G. M. Coghill. A model-based approach to robot fault diagnosis. *Knowledge Based Systems.*, 18:225–233, August 2005. ISSN 0950-7051. doi: http://dx.doi.org/10.1016/j.knosys.2004.10.004. 15

George M. Coghill Honghai Liu. A model-based approach to robot fault diagnosis. *Knowledge-Based Systems*, 2005. 14

Wolfe Jason, Bhaskara Marthi, and Stuart J. Russell. Combined task and motion planning for mobile manipulation. Technical Report UCB/EECS-2010-27, EECS Department, University of California, Berkeley, Mar 2010. 130

Michael Karg, Martin Sachenbacher, and Alexandra Kirsch. Towards expectation-based failure recognition for human robot interaction. In *Proceedings of 22nd International Workshop on Principles of Diagnosis (DX-2011).*, 2011. 15, 128

Christian Kohler, Artur Ottlik, Hans-Hellmut Nagel, and Bernhard Nebel. Qualitative reasoning feeding back into quantitative model-based tracking. Technical report, Albert-Ludwigs-Universitat, Freiburg. Germany, 2004. URL `http://cogvisys.iaks.uni-karlsruhe.de`. 130

Tom Michael Mitchell. *Machine Learning.* McGraw-Hill Series in Computer Science. WCB/McGraw-Hill, Boston, MA, 1997. 4, 5, 78, 117

A. Monteriu, P. Asthana, K. P. Valavanis, and S. Longhi. Real-time model-based fault detection and isolation for ugvs. *J. Intell. Robotics Syst.*, 56:425–439, November 2009. ISSN 0921-0296. 14

Kei Okada, Mitsuharu Kojima, Satoru Tokutsu, Yuto Mori, Toshiaki Maki, and Masayuki Inaba. Task guided attention control and visual verification in tea serving by the daily assistive humanoid hrp2jsk. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008a. 15

Kei Okada, Satoru Tokutsu, Takashi Ogura, Mitsuharu Kojima, Yuto Mori, Toshiaki Maki, and Masayuki Inaba. Scenario controller for daily assistive humanoid using visual verification. In *Proceedings of the 10th International Conference on Intelligent Autonomous Systems (IAS-10)*, 2008b. 130

A. M. Okamura, N. Smaby, and M. R. Cutkosky. An overview of dexterous manipulation. volume 1, pages 255–262 vol.1, 2000. 130

O. Pettersson, L. Karlsson, and A. Saffiotti. Model-free execution monitoring in behavior-based robotics. *IEEE Trans. on Systems, Man and Cybernetics.*, 37(4):890–901, 2007. 14

P. M. Frank R. J. Patton. *Issues for fault diagnosis of dynamic systems.* Springer-Verlag, 2000. 12, 14

David A. Randell, Zhan Cui, and Anthony G. Cohn. A spatial logic based on regions and connection. In *Proceedings of 3rd international conference on knowledge representation and reasoning*, 1992. 11

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, December 2002. ISBN 0137903952. 6

Paulo Santos and Murray Shanahan. Hypothesising object relations from image transitions. In *ECAI(2002)*, pages 292–296, 2002. 11

Russell Smith. Open dynamic engine, May 2007. URL `www.ode.org`. 22

Gerald Steinbauer. Survey on faults of robots used in robocup, 30.03.2011 2011. URL `http://www.ist.tugraz.at/rfs/`. 15

J. Tegin, J. Wikander, S. Ekvall, D. Kragic, and B. Iliev. Experience based learning and control of robotic grasping. In *Proceedings of IEEE-RAS International Conference on Humanoid Robots Workshop "Towards Cognitive Humanoid Robots" (HUMANOIDS)*, 2006. 130

Ryohei Ueda, Yohei Kakiuchi, Shunichi Nozawa, Kei Okada, and Masayuki Inaba. Anytime error recovery by integrating local and global feedback with monitoring task states. In *Proceedings of the 15th international conference on advanced robotics*, 2011. 15, 129

Vandi Verma and Reid G. Simmons. Scalable robot fault detection and identification. *Robotics and Autonomous Systems*, 54(2):184–191, 2006. 14

Vandi Verma, Geoffrey Gordon, Reid Simmons, and Sebastian Thrun. Real time fault diagnosis robot fault diagnosis. *Robotics and Automation Magazine*, 11(1):56 – 66, June 2004. 14

Phongtharin Vinayavekhin. Dexterous manipulation planning from human demonstration. Technical report, University of Tokyo, 2009. URL `citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.148.1890`. 130

D. S. Weld and J. De Kleer, editors. *Readings in Qualitative Reasoning About Physical Systems*. Morgan Kaufman, San Mateo, Ca, 1990. 15